The**Financial**Edge™

VBA and API Guide

100509

FE-VBAandAPIGuidefortheFinancialEdge-100509

VBA and API Guide for the Financial Edge

Contents

Introduction to VBA and API

Chapter 1

Contents

Welcome to the Blackbaud *VBA and API Guide* for ***The Financial Edge***. This guide provides an overview of the optional modules *VBA* and *API*, and the various tools, objects, and methods developers can use to customize Blackbaud software. Although this guide contains "introductory" information, it is not intended for use by novice users, rather it is written to introduce experienced programmers to the unique Blackbaud programming environment. Because of the complex nature of programming and the irreversible damage that can result, you should not attempt to use this information unless you are comfortable with the *Visual Basic* programming language, data types, variable scoping, and the *Visual Basic* editor.

To illustrate certain processes or procedures, each chapter of this guide contains code samples written in *Visual Basic 6.0*. This language is shared by *Visual Basic for Applications* (VBA), Microsoft *Visual Basic 6.0*, Microsoft *Office 2000*, and other VBA 6.0 host applications.While it is possible to use the *API* from other languages (C++ or Java, for example), Blackbaud provides support for only *Visual Basic* programming. Blackbaud Customer Support can help explain the intended functionality of procedures in this guide, however we will not modify, or assist you in modifying, these examples to provide additional functionality.

All programming samples are for illustration only, and are provided without warranty, either expressed or implied. This includes, but is not limited to, the implied warranties of merchantability and/or fitness for a particular purpose. To learn more about the optional modules *VBA* and *API*, contact our Sales department at sales@blackbaud.com.

The VBA and API help file provides additional resources that are not included in the print or PDF versions of this guide because they are not suited to a print format. For example, the help file contains comprehensive lists of all tables, views, and objects that appear in your Blackbaud program, and their objects, methods, primary keys, or field types. To access the VBA and API help file from the program shell, on the menu bar, select **Help**, **VBA/API Help**. You can also browse to the FESolutions.chm file in the program directory. From the VBA and API help file, you can copy sample code and paste it into *Notepad* or directly into your *VBA* or *Visual Basic 6.0* project.

Information in this guide is divided into five chapters:

Introduction. This chapter provides an overview of basic concepts, the type libraries, objects, and collections.

Programming Basics. This chapter discusses the basics of programming with *VBA* and *API*, including managing data objects, data object collections, user interface objects, service objects, interfaces, and transactions. This chapter also contains simple examples of custom modifications you can make to the software.

Blackbaud VBA. This chapter discusses using the optional module *VBA* to add customized functionality directly in the program shell.

Blackbaud API. This chapter discusses using the optional module *API* to customize the program using third-party or custom applications.

Sample Programs. This chapter contains sample applications and plug-ins, which are custom-built applet extensions you can "plug in" to the program interface to create custom solutions ranging from simple HTML documents to multi-level ActiveX documents or interactive spreadsheets.

This guide uses text formatting to identify specific items or characteristics such as programs, modules, keyboard actions, and code.

Format	Denotes
Bold Italics	This is a Blackbaud program name, such as ***General Ledger***.
Italics	This is a Blackbaud module name, such as *Configuration*.
Bold	This is the name of a field or button. For example, **Delete** refers to a button on the screen.
KEY	This is a key on your keyboard. "Press **DELETE**" refers to the key on your keyboard. On the other hand, "Click **Delete**" refers to an on-screen button.
KEY+KEY	Two keys joined by a plus sign indicate you must press the keys simultaneously to execute the command. For example, when instructed to "Press **CONTROL + F**", you press the **CONTROL** key, hold it, then press the **F** key.
KEY, KEY	Two keys separated by a comma indicate you must press the keys in the order in which they appear. For example, if you are instructed to "Press **A, F**", you must press the **A** key, release it, then press the **F** key.

Our code samples follow *Visual Basic* conventions and use green text offset by a single quotation mark to identify programmer comments. To distinguish them from standard text, code samples always appear in this format:

```
'Programming Example-
'We put VB code comments in green

Dim oFund as CGLFund
Set oFund = New CGLFund

oFund.Init FE_API.SessionContext
```

Overview of VBA and API

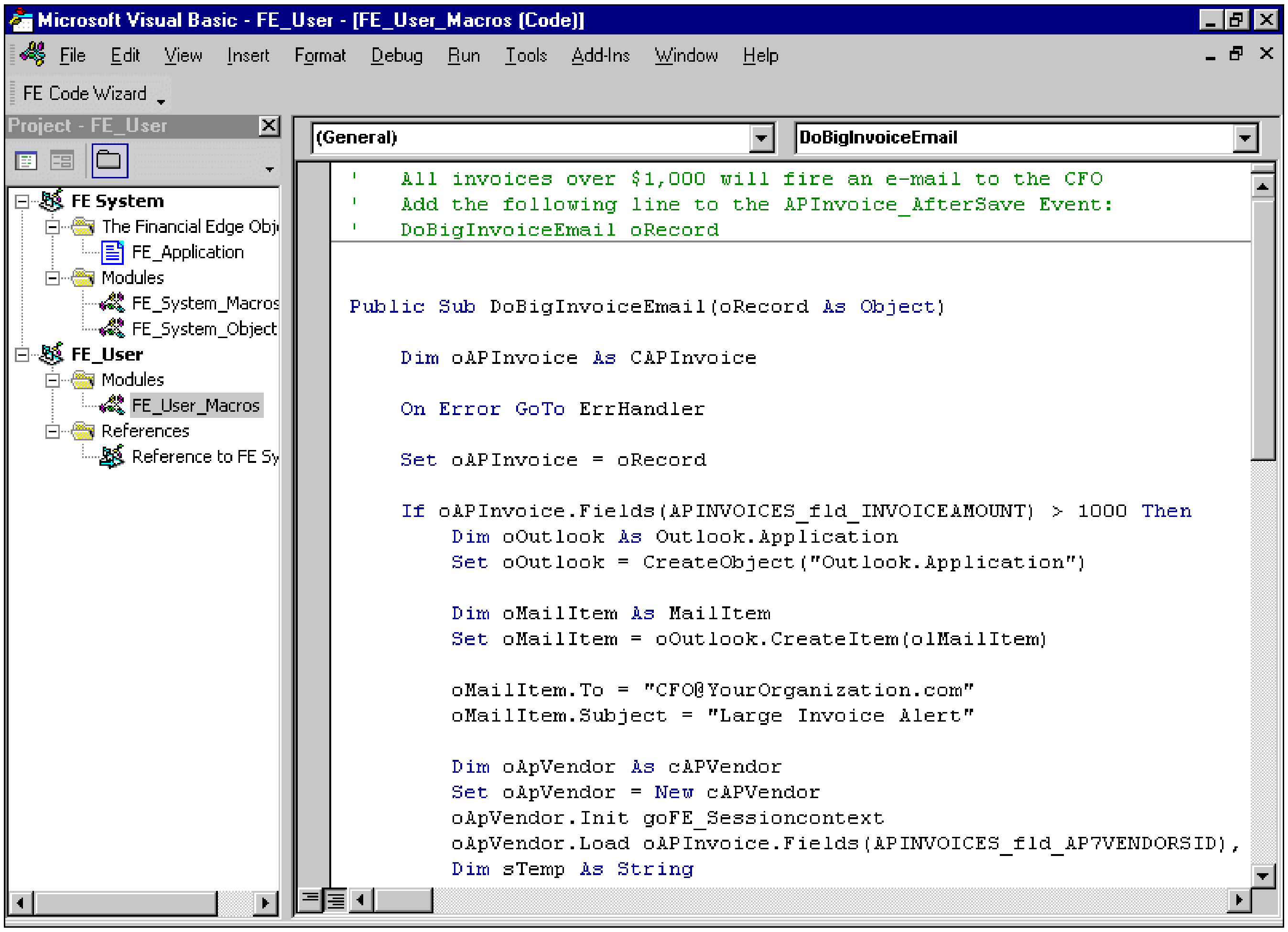
With *VBA* and *API*, you can customize Blackbaud software to meet your organization's unique needs. Whether you want to build a quick VBA macro or a full-blown application based on our program's object API, you can use these tools to accomplish your task. *VBA* and *API* have different purposes and advantages when modifying our software:

Visual Basic for Applications. *Visual Basic for Applications* is the premiere development technology for rapidly customizing and integrating packaged applications. VBA offers a sophisticated set of Microsoft *Visual Basic* programming tools you can use to create custom solutions for your specific business needs.

With VBA, you can:

- Modify application behavior. You can modify the program to match your organization's business rules and processes.
- Automate repetitive tasks. You can combine sets of common manual tasks into a repeatable series of actions.
- Extend application functionality. You can add features to the program that are not available out of the box.
- Integrate with other applications. You can adapt your Blackbaud program to work seamlessly with other VBA-enabled software to integrate a line of business applications.
- Access data. You can exchange data with remote databases and applications and deliver results directly to the desktop.

For more information about *VBA*, see “Blackbaud VBA” on page 73.



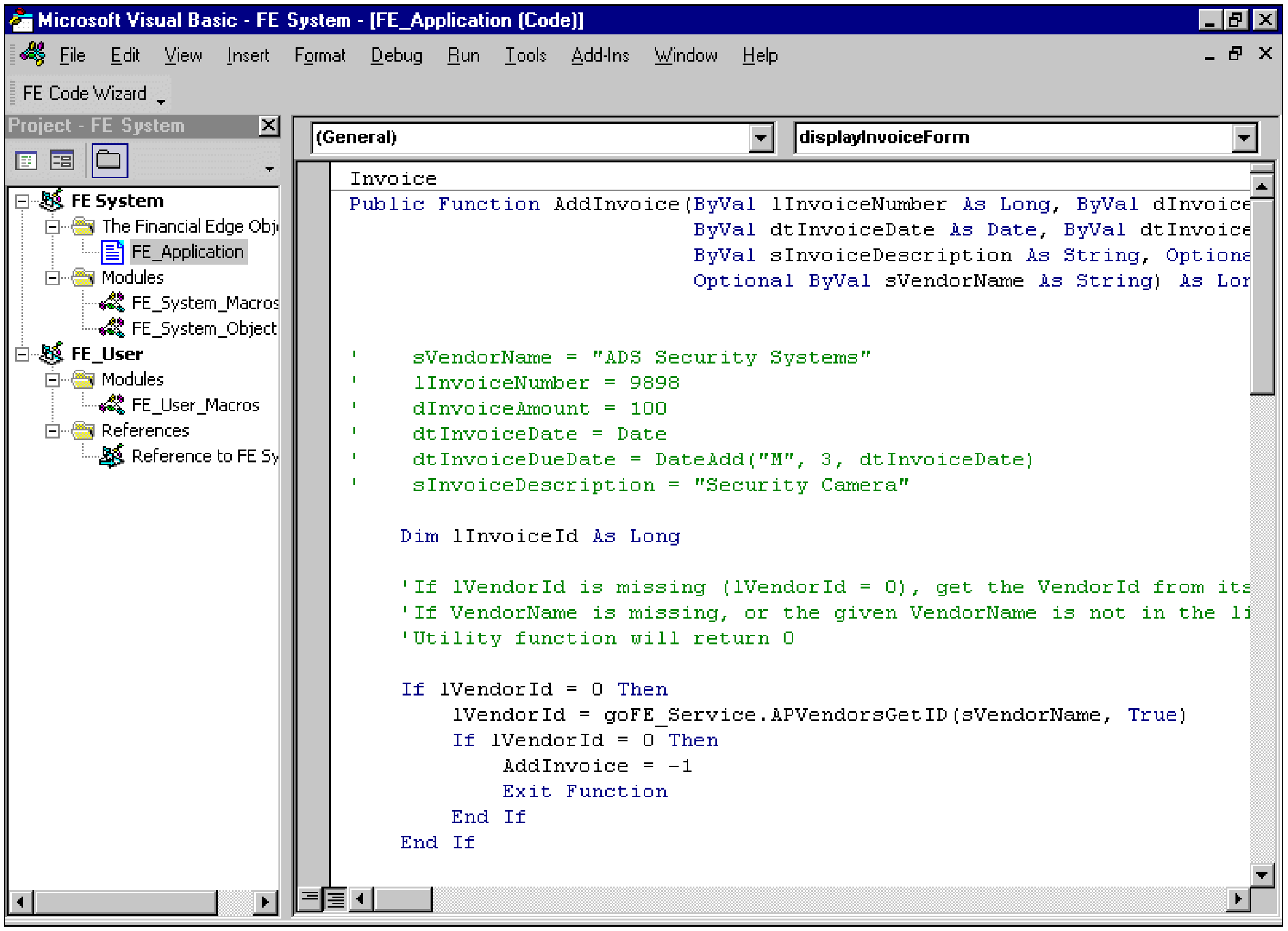
```
'   All invoices over $1,000 will fire an e-mail to the CFO
'   Add the following line to the APInvoice_AfterSave Event:
'   DoBigInvoiceEmail oRecord


Public Sub DoBigInvoiceEmail(oRecord As Object)

    Dim oAPInvoice As CAPInvoice

    On Error GoTo ErrHandler

    Set oAPInvoice = oRecord

    If oAPInvoice.Fields(APINVOICES_fld_INVOICEAMOUNT) > 1000 Then
        Dim oOutlook As Outlook.Application
        Set oOutlook = CreateObject("Outlook.Application")

        Dim oMailItem As MailItem
        Set oMailItem = oOutlook.CreateItem(olMailItem)

        oMailItem.To = "CFO@YourOrganization.com"
        oMailItem.Subject = "Large Invoice Alert"

        Dim oApVendor As cAPVendor
        Set oApVendor = New cAPVendor
        oApVendor.Init goFE_Sessioncontext
        oApVendor.Load oAPInvoice.Fields(APINVOICES_fld_AP7VENDORSID),
        Dim sTemp As String
```

Application Programming Interface. An Application Programming Interface (API) enables you to write custom applications while taking advantage of the wealth of code contained within the Blackbaud program. *API* follows the guidelines of Microsoft’s *Component Object Model* (COM), so you can use it from any COM-enabled programming environment, including Microsoft *Visual Basic*, Microsoft *Visual C++*, and Microsoft *Visual Basic for Applications*. Experienced programmers can use *API* to create applications that work with the accounting package or access Blackbaud data from almost any application.

With *API*, you can:

- Create custom form letters within Microsoft *Word* that directly access the latest information from your database.
- Generate up-to-the-minute comparative information from within Microsoft *Excel*.
- Build custom forms that aggregate the fields you use most often.
- Exchange information between your Blackbaud program and legacy systems in real time.
- Access current accounting data directly from your own Web pages.

For more information about *API*, see “Blackbaud API” on page 101.



```
 Invoice
 Public Function AddInvoice(ByVal lInvoiceNumber As Long, ByVal dInvoice
                            ByVal dtInvoiceDate As Date, ByVal dtInvoice
                            ByVal sInvoiceDescription As String, Optiona
                            Optional ByVal sVendorName As String) As Lor

 '    sVendorName = "ADS Security Systems"
 '    lInvoiceNumber = 9898
 '    dInvoiceAmount = 100
 '    dtInvoiceDate = Date
 '    dtInvoiceDueDate = DateAdd("M", 3, dtInvoiceDate)
 '    sInvoiceDescription = "Security Camera"

     Dim lInvoiceId As Long

     'If lVendorId is missing (lVendorId = 0), get the VendorId from its
     'If VendorName is missing, or the given VendorName is not in the li
     'Utility function will return 0

     If lVendorId = 0 Then
         lVendorId = goFE_Service.APVendorsGetID(sVendorName, True)
         If lVendorId = 0 Then
             AddInvoice = -1
             Exit Function
         End If
     End If
```

Comparing VBA and API

If you are not familiar with both *VBA* and *API*, you may not be sure which module is best suited to the solution you want to create. The most important distinction between *VBA* and *API* is that *VBA* is available only when your Blackbaud software is actually up and running, so *VBA* applications also only work with the program open. With *API*, you can write fully functional “standalone” programs that have complete access to Blackbaud data and services, but that can run independently of our software. *API* is the appropriate solution if you want to write your own “front-end” to the program or create a customized program that melds your accounting system with some other specialized functionality.

Programming with *API* requires you to have your own COM-enabled programming language. None of the niceties inherent to *VBA* are present in *API*. For example, *VBA* includes a complete forms design package, but *API* does not. If you want to build a user interface using *API*, you must do so on your own.

If your goal is to create an entire application or utility, *API* provides you with the perfect blend of structure and flexibility to accomplish this task. You can also use *API* to gain access from other VBA-enabled applications. For example, you can build a custom Microsoft *Excel* VBA batch entry macro for adding multiple records to ***The Financial Edge***. For more information about *VBA*, see “Blackbaud VBA” on page 73. For more information about *API*, see “Blackbaud API” on page 101.

Understanding Blackbaud Program Architecture

The program structure of ***The Financial Edge*** is unique in that programs share a common shell, the same object model, and many common objects, but the type library is split into separate object references.

To help distinguish which programs an object applies to, we added a prefix to the object name. For example, in cGLAccount, “c” indicates that the object is a class, and “GL” indicates that the object represents a ***General Ledger*** account. Common objects used in multiple programs usually contain only the “c” prefix. Examples of common objects include cBank, used in several subledger programs, and the cQueryObject interface, used in all Blackbaud accounting products.

For simplicity, most examples and code samples used in this guide are written for a particular Blackbaud program. However, you can use many samples in other Blackbaud programs by changing the objects in the code sample. To view code samples, see “Sample Programs” on page 123.

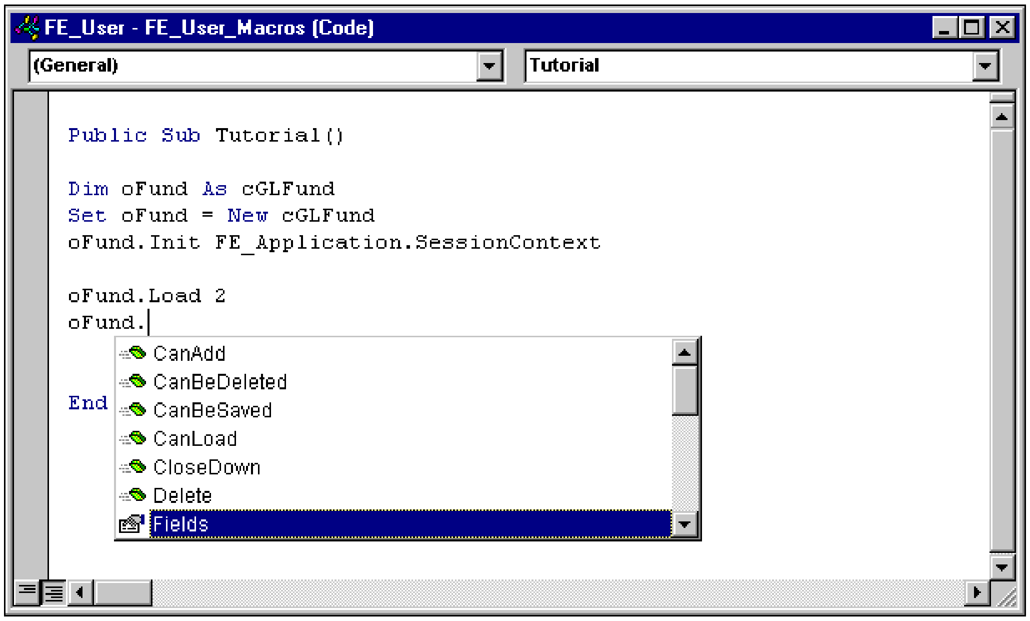
Using the Type Library

A type library is a language-independent file that provides type information about the components, interfaces, methods, constants, enumerations, and properties exposed by the system. Your code runs faster through a type library because objects are identified by specific type. This makes using the type libraries the easiest and most efficient way to manipulate Blackbaud objects. Also, if you set a reference to a type library, all objects and their properties and methods appear in the Object Browser. If you do not use a type library, *Visual Basic* must communicate with components through the slower dispatch interface. Another major drawback to using the dispatch interface method is that VBA does not provide compile-time syntax checking.

In *Visual Basic 5.0* and later, you can also use Intellisense to speed data entry. Intellisense is a *Visual Basic* feature that displays a list of an object’s properties and methods. You can then double-click the property or method and it appears in your code for faster data entry without syntax errors. For example, in *Visual Basic* or VBA, if you enter an object variable that is defined in a type library or *Visual Basic* component, then enter a period (.), the code editor displays a list of the object’s properties and methods. When you double-click a property or method in the list, it appears automatically in your code. Intellisense works only for early-bound objects. For more information about binding objects, see “Using Early-Bound Objects” on page 14.

While this guide is written for programming with *Visual Basic*, you can use the Blackbaud type library in any COM compatible language, for example Microsoft *Visual C++*.

The following picture shows the Intellisense list of properties and methods available for a fund object:



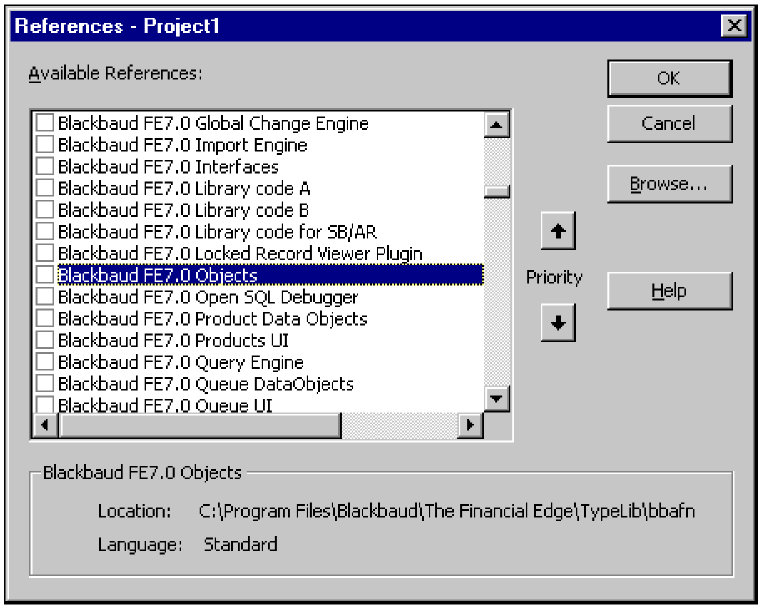
```
Public Sub Tutorial()

Dim oFund As cGLFund
Set oFund = New cGLFund
oFund.Init FE_Application.SessionContext

oFund.Load 2
oFund.

End
```

If you have the optional module *VBA*, the program automatically sets a reference to the type library when you start VBA. To manually set a reference to a library, select **Tools**, **References**.

If you have the optional module *API*, you must manually reference the type library from any *Visual Basic* project that you want to gain early-bound access to Blackbaud objects.

➢ Setting a manual reference to a type library from an API application

1. From a new project in *Visual Basic 5.0* or later, on the menu bar, select **Project**, **References**. The References screen appears, displaying a list of preset type library references.



- [ ] Blackbaud FE7.0 Global Change Engine
- [ ] Blackbaud FE7.0 Import Engine
- [ ] Blackbaud FE7.0 Interfaces
- [ ] Blackbaud FE7.0 Library code A
- [ ] Blackbaud FE7.0 Library code B
- [ ] Blackbaud FE7.0 Library code for SB/AR
- [ ] Blackbaud FE7.0 Locked Record Viewer Plugin
- [ ] Blackbaud FE7.0 Objects
- [ ] Blackbaud FE7.0 Open SQL Debugger
- [ ] Blackbaud FE7.0 Product Data Objects
- [ ] Blackbaud FE7.0 Products UI
- [ ] Blackbaud FE7.0 Query Engine
- [ ] Blackbaud FE7.0 Queue DataObjects
- [ ] Blackbaud FE7.0 Queue UI

Blackbaud FE7.0 Objects

Location: C:\Program Files\Blackbaud\The Financial Edge\TypeLib\bbafn

Language: Standard

2. In the list, mark **Blackbaud FE7.0 Objects**. The Blackbaud FE7.0 reference must be set for you to gain early-bound access to Blackbaud objects.

3. To close the References screen and return to the project, click **OK**.

Understanding Objects, Object Models, and Collections

Before you can begin programming, you must understand objects. Objects are the basic building blocks of any application, including ***The Financial Edge***. In fact, nearly everything you do in *Visual Basic* involves manipulating objects. Once you understand how to work with objects, you can successfully customize our software.

An object is code and data combined into a single unit that can be as simple as a single piece of an application or as complex as an entire application. In Blackbaud software, every data element — each account, project, and invoice — is an object you can manipulate programmatically in *Visual Basic*. Examples of content-type objects containing data are invoices, accounts, account codes, vendors, and projects. Objects also include functionality-based, or “service”, objects that manipulate content by opening, closing, adding, displaying, finding, or deleting records. Service objects include queries, reports, and viewers.

For a complete list of programmable objects in ***The Financial Edge***, see the Programmer Reference section of the VBA and API help file. To access the VBA and API help file from the program shell, select **Help**, **VBA/API Help**.

Understanding Object Models

An object model is a single structure created from many objects. Further, groups of object models combine to create applications such as ***The Financial Edge***. An object model is similar to a family tree in that it contains levels of parents and children. In an object model, the parents are known as top-level objects, and the children are called child objects. As you progress from upper to lower levels of the object model, child objects can have their own children, and the child objects “inherit” certain characteristics from their parents.

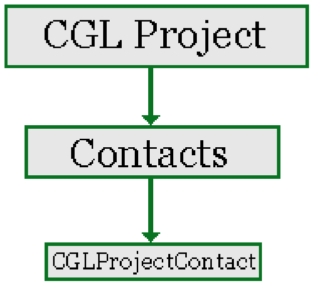
In a human family tree, individuals within the family structure are physical beings, but they are also associated with what they do (doctor, student, mother, etc.). Similarly, *Visual Basic* objects can be content such as invoices, accounts, account codes, vendors, projects, and they can represent functionality, reacting to various situations or events. Examples of functionality-based objects include Init, CloseDown, Load, or Delete objects that perform actions on records.

In Blackbaud programs, each major record type has its own object model. For example, a top-level object such as a project has a model comprised of child objects such as contacts, attributes, and notepads. If you build object models with the same layout as their Blackbaud counterparts, the task of programming such a large and complicated relational database is greatly simplified. The Blackbaud object model has one major goal: to expose all important functionality and data needed to manipulate the database records and services in a high-level manner.

Understanding Data Objects

The Blackbaud object model is primarily based around the data the program manages. Because the key to your accounting system is its data, data objects are the key to programming ***The Financial Edge***. Higher level data objects are called “top-level” objects, and lower level data objects are considered “child” objects. To illustrate this relationship, in ***General Ledger***, project records can contain contacts that track biographical information about individuals related to the project. A project can have any number of contacts on its record. In relational database terms, there is said to be a “one-to-many” relationship between a project and its contacts. For this reason, in ***The Financial Edge*** object model, the contacts object is a child of the CGLProject data object, which is the object in the system that represents projects.

In the following diagram, we see that for each CGLProject there is a child object named Contacts, and the Contacts object has child object named CGLProjectContact. The Contacts object name is plural for a very important reason — it is a collection object used to iterate through any number of children.



You can navigate all collection objects in the object model using the “For Each” syntax, which is the standard for navigating in VBA collections. The following code sample illustrates navigating a project object using the “For Each” syntax:

```
'Note: The code to initialize and load a CGLProject(oProject)
'    object omitted for brevity
Dim oProjectContact as CGLProjectContact

'Print all of this project's contact names to the
'    VBA debug window
For Each oProjectContact in oProject.Contacts
    Debug.Print oProjectContact.Fields(GL7PROJECTCONTACTS_fld_NAMEID)
Next oProjectContact
```

Every data object in the program is modeled in the same manner. After you become familiar with the “blueprint” for the Blackbaud object model, you can program any data object in the program.

Understanding Top-Level Objects

A top-level object gets its name because it is at the top of the object model hierarchy. A method is the programming equivalent of a verb — it performs an action or service for objects in the program. The methods of a top object provide access to other objects and collections. All top-level objects have the same methods.

Top-Level Object Methods and Properties:

Method	Description
Init	Must be called before using the object; must pass in an IBBSessionContext
Load	Pass in the ID of the object to load it from the database
Fields	Allows you to change the value of any of the fields for the object
Save	Saves any changes you made to the objects fields or other objects in its hierarchy to the database
CloseDown	Must be called when you are through with the object
Delete	Deletes the object and any related records from the database

Understanding Child Objects

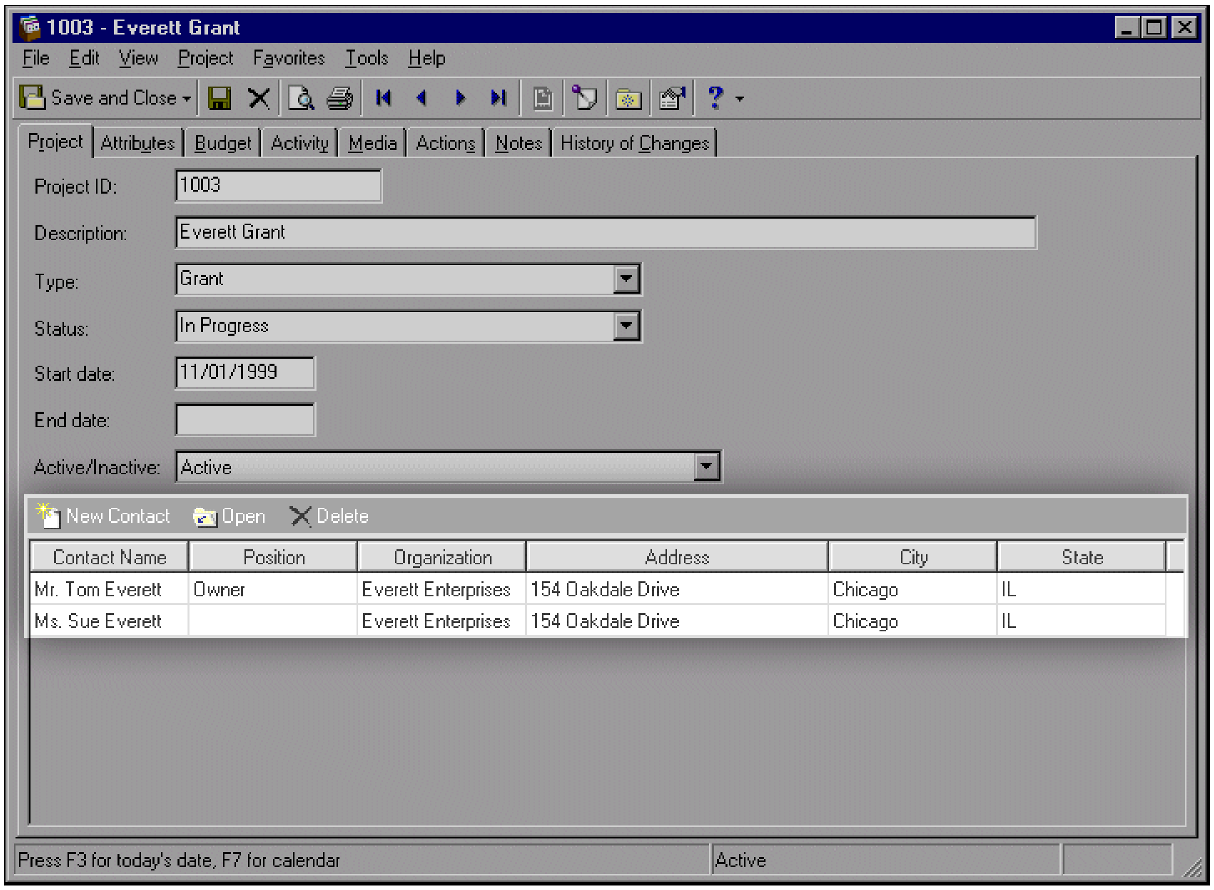
Child objects are lower-level objects within the object model hierarchy. Attributes, notes, and the history of changes are common examples of child objects. A child object cannot exist without a top-level object. For example, to add a contact to a project in ***General Ledger***, you must first load the project. To add contacts programmatically, you must also first load and initialize the parent record. You cannot create, load, save, initialize, or delete child objects. All these actions are accomplished via methods exposed by the child object's parent. Child objects contain no common methods, but they share a common property, the Fields property.

Child Object Properties:

Property	Description
Fields	Enables you to change the value of any of the object's fields

A ***General Ledger*** project form includes several parent-child relationships, such as that of the project and its contacts. A project's contacts are children of the record and are available only through the project's CGLProject object.

The following picture shows contact child objects appearing on a project record:


1003 - Everett Grant
File Edit View Project Favorites Tools Help
Save and Close
Project Attributes Budget Activity Media Actions Notes History of Changes
New Contact Open Delete

Project ID: 1003
Description: Everett Grant
Type: Grant
Status: In Progress
Start date: 11/01/1999
End date:
Active/Inactive: Active

Contact Name	Position	Organization	Address	City	State
Mr. Tom Everett	Owner	Everett Enterprises	154 Oakdale Drive	Chicago	IL
Ms. Sue Everett		Everett Enterprises	154 Oakdale Drive	Chicago	IL

Press F3 for today's date, F7 for calendar Active

Understanding Object Collections

A collection is a parent object containing other objects that are related to each other. Using collections provides a simple way to group objects into a single unit you can refer to collectively rather than by identifying each piece. You create collections the same way you create other objects. For example, to create an account collection, use this format:

```
Dim oAccounts As CGLAccounts
Set oAccounts = New CGLAccounts
```

Understanding Top-Level Collections

The top-level collection is a collection of top-level objects. These objects are useful if you want to process all the instances of a given object in the program. Top view collections have no add or remove methods because you add and remove top-level data collections through methods on the top-level objects themselves. A powerful feature of top view collections is that you can apply a filter to a collection when it is initialized so that only a specific subset of objects is included. For example, you may want to only include active accounts when using the CGLAccounts collection. In this case, you pass the correct filter constant (tvf_Account_CustomWhereClause) and enter the custom filter criteria as a string. When the collection is initialized, it includes only accounts marked as active. This additional parameter is optional. You should reference the database tables in the VBA and API help file to find the correct “where clause”.

In collections, the object name becomes plural.

Top-Level Collection Methods:

Method	Description
CloseDown	Must be called when you are through with the object
Count	Provides a count of the number of objects in the collection
Init	Must be called before using the object. Must pass in an IBBSessionContext
Item	Returns a top object for a given index

This code sample illustrates filtering the CGLAccounts collection for active accounts.

```
'Define a variable to navigate the top view collection

Dim oAllAccounts as CGLAccounts
Dim oAccount as CGLAccount

Set oAllAccounts = new CGLAccounts

oAllAccounts.Init FE_Application.SessionContext, _
    "Status = 1", , tvf_Account_CustomWhereClause

For Each oAccount in oAllAccounts
    Debug.Print oAccount.Fields(GLACCOUNTS_fld_DESCRIPTION)
oAccount.CloseDown

Next oAccount
```

Each top-level object has a corresponding top view collection. Remember, a distinguishing characteristic of a collection is that the object’s name takes the plural form. For example, the top view collection object for CGLAccount is CGLAccounts.

The following example illustrates filtering undeleted invoices and printing a list by the invoice description:

```
Dim oInvoice As CAPInvoice
Dim oInvoices As CAPInvoices

Set oInvoices = New CAPInvoices

oInvoices.Init FE_Application.SessionContext, tvf_Invoice_UseFilterObject
oInvoices.FilterObject.ExcludeDeleted = True

For Each oInvoice In oInvoices
    Debug.Print oInvoice.Fields(APINVOICES_fld_DESCRIPTION)
    oInvoice.CloseDown
    ' close down top object
Next
Set oInvoice = Nothing

oInvoices.CloseDown
Set oInvoices = Nothing
```

Understanding Child Collections

A child collection is a collection of child objects. A child collection cannot exist without a top object. You can add to or remove objects from the collection, but you cannot save a child object without calling the parent's Save method. This is because the parent may have to enforce rules about membership in its collection. When you save the parent, all the child objects in the collection are saved if they are dirty (they have been edited) or are new, and all objects that have been removed are deleted from the database. You cannot create, load, save, close down, initialize, or delete a child collection.

Child Collection Methods:

Method	Description
Item	Returns a child object, given an index.
Add	Creates a new child object, stores its membership in the collection, and returns a reference to it.
Remove	Removes a child object from the collection. Once a child object is removed from a collection, it cannot be used again.
Count	Provides a count of the number of child objects in the collection.

Understanding Child View Collections

With a child view collection, you can navigate through a subset of a particular true Child collection. You cannot add to or remove from these collections because they are just views of another collection and other factors determine their membership. Child view collections have two methods.

Child View Collection Methods:

Method	Description
Item	Returns a child object given an index.
Count	Provides a count of the number of child objects in the collection.

Understanding Service Objects

While data objects enable you to manipulate data within your program, and user interface objects give you a workable form for interacting with the user, other objects give you access to discrete functionality within the application. These objects are not easily categorized because they each provide a service via their own unique programming interfaces. To help organize these entities, Blackbaud's object model refers to them as Service Objects. Service objects include, but are not limited to, queries, reports, viewers, search screens, and some forms. It is likely you will call upon service objects quite frequently as you tackle various development tasks with the program. For example, with the Query service object, you can access the result set of pre-existing queries to improve or expand the program's reporting and data analysis capabilities. Another advantage of service objects is that they enable you to quickly assemble solutions that leverage existing Blackbaud functionality, while at the same time presenting users with a familiar interface.

For more information about service objects, see "Programming Basics" on page 19.

Working with Objects

Working with objects is the basis for programming. In both *VBA* and *API*, you should perform certain operations or follow specific procedures when working with objects, including using early-bound objects from the type libraries, and initializing and releasing objects. Failure to follow these processes can lead to syntax problems and run-time errors.

Using Early-Bound Objects

The Blackbaud type libraries provides the declarations required by *Visual Basic* and VBA for every documented program object, method, and constant. We strongly recommend you use early-bound, strictly-typed variables from the type libraries.

The following code sample illustrates the advantage of using early-bound objects:

```
'This variable is late-bound. While it will still work,
'    it will incur significant run time overhead, and will not yield Intellisense.

Dim oFund As Object
Set oFund = New CGLFund

'This early-bound variable provides optimal speed and
'    access to the VB/VBA intellisense feature.

Dim oFundEarly As CGLFund
Set oFundEarly = New CGLFund
```

Initializing and Releasing Objects

Whenever you use an object exposed by the object model, it must first be initialized. In Blackbaud software, the key to programming is the SessionContext object parameter. This object holds information about the state of the active instance of the application.

When you create new instances of objects and initialize them with a SessionContext, the object queries the SessionContext for information it needs to operate (for example, a handle to the low-level database connection interface).

To properly initialize an object, you must pass a reference to the SessionContext. Almost every top-level object in ***The Financial Edge*** is initialized using the same method. You initialize (.Init) with a SessionContext, then release (.CloseDown) the object when you are done. If you attempt to use an object without properly initializing it, a trappable run-time error occurs. The SessionContext is obtained in slightly different ways, depending on whether you are using the VBA or API development platform.

Initializing Objects in VBA

In VBA, the SessionContext is exposed via the FE_Application object. The FE_Application object is a global object available to VBA. The most important property on the FE_Application object is the SessionContext. The following code sample illustrates initializing a CGLAccount object in VBA:

```
Dim oAccount as CGLAccount
Set oAccount = New CGLAccount

'Use the FE_Application object to get a reference to the
'SessionContext
oAccount.Init FE_Application.SessionContext

'Load Account with Database ID of 1.
oAccount.Load 1

'Release reference to GLAccount Object
oAccount.CloseDown
```

Initializing Objects in API

In *API*, you also must initialize objects before using them. It is important to understand that while a few differences exist, once you understand object programming, the same rules apply to both *VBA* and the *API*.

An API application obtains its reference to the SessionContext via the FEAPI object. Unlike the FE_Application object, which is automatically initialized and available to VBA in the running instance of your Blackbaud application, you must manually initialize API using the object’s Init method. With the Init method, you can then log into the program.

The following table lists parts of the FE_API.Init method and their descriptions.

Part	Description
sSerialNumber	Required. You don’t have to enter a serial number, but the parameter is required. Use double quotes " ".
sUserName	Optional. A string expression containing a valid user name for the database to which you are attempting to connect. This appears on the login form. If the user name and password fields are left blank, the login form appears when the Init method runs.
sPwd	Optional. A string expression containing a valid password for the user name specified above. If both user name and password are supplied, the login form is not displayed.
DatabaseNumber	Optional. A long expression representing the position of the desired database within the login list. Note the standard sample database is always represented by 50. The first live database is usually represented by 1. If the optional parameter is not defined, the database form appears to the user, enabling them to select the desired database.
sThirdPartyVendor	Optional. Reserved for third party vendors.
lAppMode	Optional. A long expression indicating whether this application is operating standalone or as a server. If omitted, FE assumes this is a standalone application.

The following code sample illustrates initializing a CGLAccount object in API:

```
Dim oAPI as FE_API

'Initialize the API and log in
Set oAPI = New FE_API

'Log in as user Bob with password "Admin"
oAPI.Init "", "Bob", "Admin"

Dim oAccount as CGLAccount
Set oAccount = New CGLAccount

'Use the API object to get a reference to the SessionContext
oAccount.Init oAPI.SessionContext

'Load GLAccount 1
oAccount.Load 1

'Release reference to GLAccount Object
oAccount.CloseDown
Set oAccount = nothing
```

The first three lines of this code remain constant for any API application and are usually placed in a section of your API applications that is executed only once, for example, in your main form's Load event.

The following code sample illustrates creating an FE_API object reference from *Visual Basic*:

```
'Create a new FE_API object and set a modular reference to it
Dim moFE_API as FE_API
Set moFE_API = New FE_API
```

The following code illustrates connecting to the sample database as "Supervisor" using the default password "Admin":

```
'Initialize the FE_API object and attempt to connect to the FE_7 sample database
If Not moFE_API.Init("", "Supervisor", "Admin", 50) Then
    MsgBox "Cannot connect to database", vbOKOnly Or vbInformation
    Exit Sub
End If
```

Releasing Objects

Closing down objects can be a little trickier than initializing them. If you fail to properly close down an object, potentially all of the object's resources will remain "alive" and in memory. To many developers, this is known as a "memory leak". The objects attempt to detect this situation and raise errors in many situations if a CloseDown call is not made. In some cases this type of leak cannot be detected immediately, leading to some hard-to-track bugs. Remember, if it has an Init method, it probably has a CloseDown method also, and you should always make sure you call them both.

The VBA code sample is representative of almost every sample of programming code you see in our accounting products:

```
Dim oAccount as CGLAccount
Set oAccount = New CGLAccount

'Initialize the oAccount via the Init method
oAccount.Init FE_Application.SessionContext

'Load Account with Database ID of 1
oAccount.Load 1

'Properly release reference to Account Object using the
'CloseDown method
oAccount.CloseDown
```

Using Foreign Keys

Foreign keys are the links between two related tables. The foreign key in a “foreign” table contains a value corresponding to the primary key of a “primary” table, ensuring that the information you add in the foreign table meets the same requirements as the corresponding data in the primary table. A relationship between primary keys and foreign keys takes one of two main forms — one-to-many or one-to-one. A good example of a one-to-many relationship is the parent/child relationship of data objects. A parent object may have many child objects, and each child contains a foreign key to its parent’s table. For example, the Project Contacts table has a field named GL7ProjectsID that relates the contact to its parent project. In order to maintain integrity in the database, the GL7ProjectsID is set by the Contacts collection on save, and it cannot be reset. This is a one-to-many relationship, because each project can contain many contacts. You can also see this relationship through the object model:

```
Dim oContact As CGLProjectContact
oContact.Fields(GLPROJECTCONTACTS_fld_GL7PROJECTSID)
```

An example of a one-to-one relationship is the Name table. Each project contact can possess only one name. Note there is no Name collection on the oContact object. Instead, there is just a single Name object. You can see the one-to-one relationship in the object model:

```
oContact.NameObject.Fields(NAME_fld_FULLNAME)
```

The Name table does not have a foreign key to the Project Contacts table, but rather the Project Contacts table has a foreign key to the Names table.

```
oContact.Fields(GLPROJECTCONTACTS_fld_NAMEID)
```

In most cases, when you access a foreign key, the program returns the automatically generated ID number that was created on the primary object. In some cases, however, the program returns a string — usually the description of the object. For example, if you access an account field, most Blackbaud objects expose the user-defined ID, such as 01-1000-00, rather than the database ID. This treatment of primary keys holds true for account and project names, and in some cases, user names. For example, if you create an action and assign it to a user, the program uses the name string instead of the user ID. You can use the IBBMetaField_FormatDescriptor to verify the format to use. For more information about the IBBMetaField interface, see the “Object Reference” section of the VBA and API help file.

The following code sample illustrates foreign keys accessed by both a user name and user ID:

```
Dim oVendor As cAPVendor
Dim oAction As CAPVendorAction
Dim oMeta As IBBMetaField

Set oVendor = New cAPVendor
oVendor.Init moSC
oVendor.Load 1

Set oAction = oVendor.Actions.Add

'Here are two examples of a foreign key to the Users table. One is set by the
'user’s name, and the other is set by the user's ID number.

'This is an example of a field that is a User Name. It returns the Name
'rather than the ID.
If oAction.Fields(ACTIONS_fld_ASSIGNEDTOID) = "Supervisor" Then
    Debug.Print "this is Supervisor"
End If

'However, the Added by field returns the UsersID value.
If oAction.Fields(ACTIONS_fld_ADDEDBYID) = 1 Then
    Debug.Print "this is also the Supervisor"
End If
```

If you are unsure whether to use the ID or the user name, you can check the FormatDescriptor on the MetaField:

```
Set oMeta = oAction
If oMeta.FormatDescriptor(ACTIONS_fld_ASSIGNEDTOID) = fmtUSER_NAME Then
    Debug.Print "This is a User Name field"
End If

If oMeta.FormatDescriptor(ACTIONS_fld_ADDEDBYID) = fmtNUMBER Then
    Debug.Print "This is a User ID field"
End If

'fmtACCOUNT_ID, fmtPROJECT_ID and fmtUSER_NAME all return and are set by the
'User defined ID field, rather than the database internal ID field.

Set oMeta = Nothing
Set oAction = Nothing

oVendor.CloseDown
Set oVendor = Nothing
```

Programming Basics

Chapter 2

Contents

This chapter introduces the basics of programming with *VBA* and *API* and provides details of working with objects and object collections.

Managing Data Objects

Data objects are an integral part of programming in Blackbaud products. To build successful applications using *VBA* or *API*, you must understand the basics of programming with data objects, including managing top-level and child objects and collections, loading, updating, and validating objects, and how to handle errors. For an overview of objects, collections, and the object model, see “Understanding Objects, Object Models, and Collections” on page 8.

Managing Top-Level Objects

To manipulate a data record in your accounting system, you must initialize and load the appropriate data object. The object model provides a data object for every editable record in your system, but only a select few data objects can be instantiated and loaded. Most data objects are “children” of another object in the hierarchy. For example, your database may have a project that contains a contact. The contact is of no use unless you know which project the contact is associated with. For this reason, a contact can not be accessed directly, but must be accessed through its top-level object — the project record. The contact object is accessed as a “child” of the project object.

Understanding the parent-child data object relationship is a key concept to grasp as you move forward with data object programming. Throughout this guide, you will see objects that are at the top of the object hierarchy referred to as “top-level objects”, and any objects that are accessible only via a top-level object are referred to as “child” objects. For an introduction to top-level and child objects, see “Understanding Top-Level Objects” on page 9 and “Understanding Child Objects” on page 10.

To view top-level objects for ***The Financial Edge*** programs, see “Financial Edge Top-Level Objects” on page 20.

Financial Edge Top-Level Objects

All ***Financial Edge*** programs, including ***General Ledger***, ***Accounts Payable***, ***Accounts Receivable***, ***Cash Receipts***, and ***Fixed Assets***, contain top-level objects.

General Ledger Top-Level Data Objects

General Ledger Top-Level Data Objects	Available with Optional Module
CGlAccount	N/A
CGLAccountCode	N/A
CGLAllocationSet	*Allocation Management*
CGLBatch	N/A
CGLBudgetDistribution	*Budget Management*
CGLBudgetScenario	*Budget Management*
CGLChartTemplate	N/A
CGLConsImport	*Consolidation Management*
CGLConsMap	*Consolidation Management*
CGLCurrencyExchange	*Consolidation Management*

General Ledger Top-Level Data Objects	Available with Optional Module
CGLDefTranDist	N/A
CGLFeeSchedule	*Allocation Management*
CGLFiscalYear	N/A
CGLFund	N/A
CGLPool	*Allocation Management*
CGLProject	*Projects and Grants*
CGLProjectAction	*Projects and Grants*
CGLRate	*Allocation Management*
CGLRecurringBatch	N/A
CGLRevaluationSet	*Consolidation Management*
CGLSegments	*Budget Management*
CGLSummary	N/A
CGLTransaction	N/A
CGLTransactionCodes	N/A

Accounts Payable Top-Level Data Objects

Accounts Payable Top-Level Data Objects	Available with Optional Module
CAPCheck	N/A
CAPCreditMemo	N/A
CAPBillItemHeader	N/A
CAPInvoice	N/A
CAPMiscLineItem	N/A
CAPOrgAddress	N/A
CAPPostInfo	N/A
CAPPostingInformation	N/A
CAPPurchaseOrder	*Purchase Orders*
CAPReceipt	*Purchase Orders*
CAPRecurringInvoice	N/A
CAPSalesTaxItem	N/A
CAPTerm	N/A
CAPVendor	N/A
CAPVendorAction	N/A

Accounts Receivable Top-Level Data Objects

Accounts Receivable Top-Level Data Objects	Available with Optional Module
CARCharge	N/A
CARClient	N/A
CARClientAction	N/A
CARCredit	N/A
CARInvoice	N/A
CARLineItem	N/A
CARRecurringInvoice	N/A
CARRefund	N/A

Accounts Receivable Top-Level Data Objects	Available with Optional Module
CARReturn	N/A
CARReturnLineItem	N/A
CARTerm	N/A

Fixed Assets Top-Level Data Objects

Fixed Assets Top-Level Data Objects	Available with Optional Module
CFAAsset	N/A
CFAAssetAction	N/A
CFAAssetClass	N/A
CFAAssetInventory	N/A
CFACustomDepSchedule	N/A
CFADepreciationYear	N/A
CFAPostinfo	N/A
CFATransaction	N/A

Common Top-Level Data Objects

Common top-level data objects are shared by multiple applications. The Report object, for example, is available in all Blackbaud programs.

These top-level data objects are available in more than one Blackbaud program:

Common Top-Level Data Objects/ID	Available with Optional Module
CAccrueAttendance	N/A
CAdjustment	N/A
CAttributeTypes	N/A
CBank	N/A
CBillingItem	N/A
CBRTransaction	N/A
CCodeTable	N/A
CCountry	N/A
CDefaultAcctsGroup	N/A
CDeposit	N/A
CExport	N/A
CInterfundSet	N/A
CPaymentHeader	N/A
CPaymentRun	N/A
CPostparameter	N/A
CProduct	*Purchase Orders*
CQueryObject	N/A
CRequisition	N/A
CSalutation	N/A
CSalutationField	N/A
CSignature	N/A
CSysBusRuleDetail	N/A
CTableEntry	N/A

Loading Top-Level Data Objects

To use objects in code, you have to “load” them. Each data object supports various methods for loading. In *Visual Basic*, you can load objects by using the database ID (a primary key) and the Load method, you can load objects using search screen, or for some Blackbaud accounting programs, you can select objects in Intellisense to speed code entry and eliminate syntax errors.

Loading Data Objects by Database ID

Each record in your accounting system is stored in the database. To define database relationships and integrity, the records are assigned unique values by the Database Management System. These values are called “primary keys”. You can load each top-level data object using this key value with the “Load” method. The load method accepts just one argument, a long integer representing the primary key of the record that you want to load.

The following code sample shows the various ways to load an account data object for account number “01-1100”. The database ID (primary key) for this account number is 1.

```
Dim oAccount as CGLAccount
Set oAccount = New CGLAccount

oAccount.Init FE_Application.SessionContext
'Load the record via the account number
oAccount.LoadByField uf_Account_AccountNumber, "01-1100"
'Load the record via the Database ID
oAccount.Load 1
```

Loading Data Objects Using the Search Screen

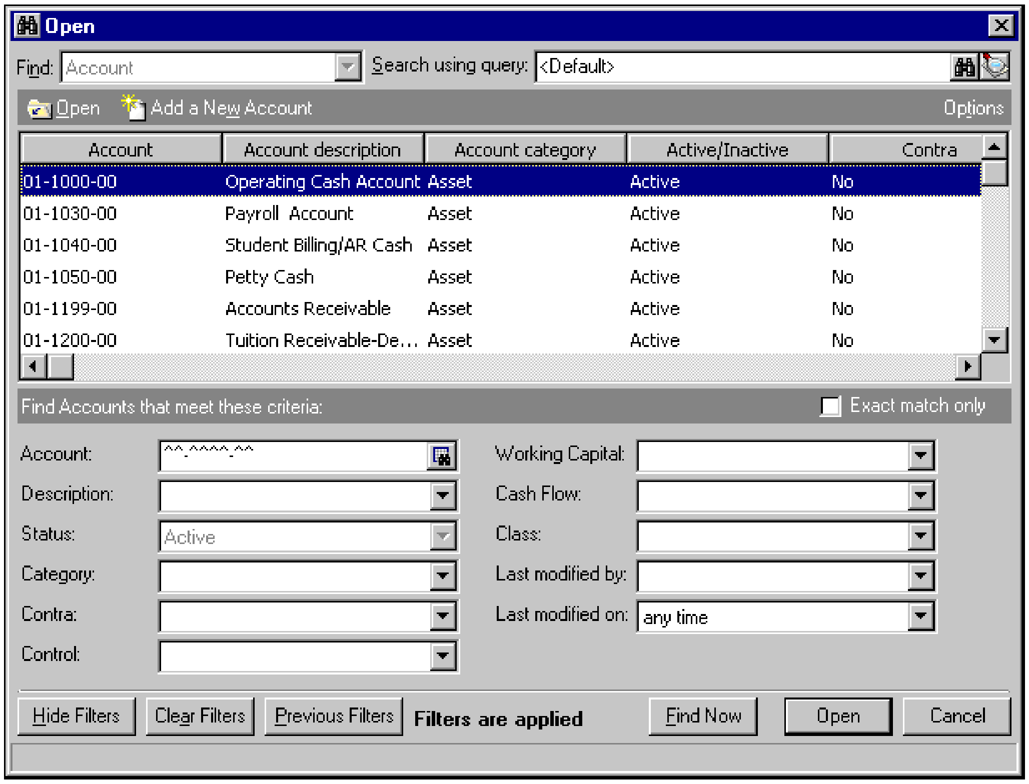
If you are building a custom search screen, loading objects by database ID is an acceptable solution. However, if you require a more robust search, or to concentrate on your application and use as many pre-built components as possible, you may want to load objects using ***The Financial Edge*** search screen.

The search screen is programmable and easily modified in *Visual Basic* code. In fact, the search screen is a “service object”, which means it is an object that provides easy access to Blackbaud’s functionality. For more information about service objects, see “Managing Service Objects” on page 43.

The following code sample shows how to load an account record using the search screen.

```
Dim oAccount as CGLAccount
Set oAccount = New CGLAccount

'Access the SearchScreen service object
Dim oServices as FE_Services
Set oServices = New FE_Services
oServices.Init FE_Application.SessionContext

'Declare variable used to access the Search Screen
Dim oSearch As IBBSearchScreen

'The services object exposes most common, useful interface
'dialogs
Set oSearch = moServices.CreateServiceObject(bbsoSearchScreen)
oSearch.Init FE_Application.SessionContext

'"Tell" the search dialog to allow for an account search
oSearch.AddSearchType SEARCH_GLACCOUNT

'Show The Search form
oSearch.ShowSearchForm

'If the user didn't cancel, assign the
'record they selected to our data object

If Not oSearch.SelectedDataObject Is Nothing Then
    Set oAccount = oSearch.SelectedDataObject
End If
```

Using this code sample, the Open screen, or “search” screen appears. If the user selects a record, the search service constructs the appropriate data object, which is accessed from code via the “SelectedDataObject” property.



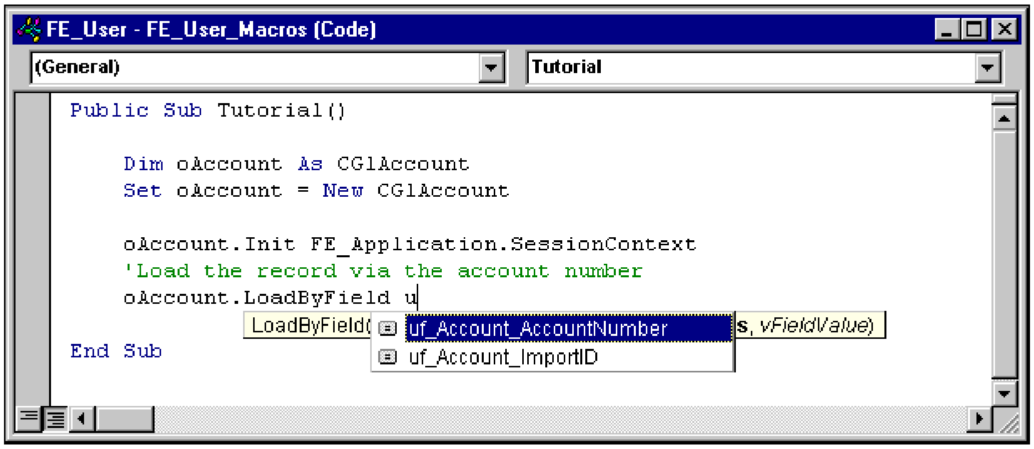
Account	Account description	Account category	Active/Inactive	Contra
01-1000-00	Operating Cash Account	Asset	Active	No
01-1030-00	Payroll Account	Asset	Active	No
01-1040-00	Student Billing/AR Cash	Asset	Active	No
01-1050-00	Petty Cash	Asset	Active	No
01-1199-00	Accounts Receivable	Asset	Active	No
01-1200-00	Tuition Receivable-De...	Asset	Active	No

Account: ^^-^^^^-^^
Description:
Status: Active
Category:
Contra:
Control:
Working Capital:
Cash Flow:
Class:
Last modified by:
Last modified on: any time

Loading Data Objects using Intellisense

In some Blackbaud accounting programs, you can load top-level objects using any of their unique fields. For example, in *General Ledger*, you cannot save two accounts with the same account number or two projects with the same project ID. Given this limitation, you can load each data object with the underlying record’s unique fields by using the LoadByField method. The LoadByField accepts two arguments. The first argument denotes the unique field you are using, and the second provides the key you want to find. In *Visual Basic*, when you fill in the first argument for the LoadByField, Intellisense displays the object’s corresponding LoadByField fields.

This picture shows the Intellisense list of constants for the LoadByField method.



```
Public Sub Tutorial()

    Dim oAccount As CGlAccount
    Set oAccount = New CGlAccount

    oAccount.Init FE_Application.SessionContext
    'Load the record via the account number
    oAccount.LoadByField u
                 LoadByField(  uf_Account_AccountNumber   s, vFieldValue)
                               uf_Account_ImportID
End Sub
```

Intellisense speeds data entry and eliminates syntax errors by providing a list of methods and properties. For more information about Intellisense, see “Using the Type Library” on page 6.

Adding Records Using Data Objects

Normally if an end user wants to add a project through *General Ledger*, he or she opens the *Records* page and clicks **New Project** to access the New Project screen. With VBA, you can also enter new records using code. Because ***Financial Edge*** records are top-level objects, they can all be entered via code, provided you define all required fields for that record type. If a required field is not defined and an error trap is not set, the application aborts when the save method is called. For more information about handling errors, see “Handling Data Object Errors” on page 30.

All top-level objects are added the same way, but because you can access child objects only via a parent object, child objects require different procedures. For more information about adding child objects, see “Adding a Child Object” on page 27.

➢ Adding a record using a data object

1. To determine required fields for a specific record type, in ***The Financial Edge***, open a new record of that type. If you have not altered the program’s default color options, required fields appear in cyan. To set color options, from the menu bar, select **Tools**, **Options**, then select the Color tab.

   The following code sample illustrates entering new project records using code. You can add any record type using the following sample. Simply change the object and define all required fields.

```
'Create a new instance of the CGLProject object
Dim oProject As CGLProject
Set oProject = New CGLProject

'Initialize the object by passing in a valid SessionContext
oProject.Init FE_Application.SessionContext

'Set any values and save
oProject.Fields(GLPROJECTS_fld_PROJECTID) = "2000"
oProject.Fields(GLPROJECTS_fld_DESCRIPTION) = "New Project"
oProject.Fields(GLPROJECTS_fld_TYPE) = "Grant"
oProject.Fields(GLPROJECTS_fld_STATUS) = "Open"
oProject.Save

'Always clean up. Objects with an Init() method typically
'have a CloseDown() method.
oProject.CloseDown
set oProject = Nothing
```

2. If required fields contain a default field entry, you do not have to define the field. For example, if you have set the **Automatically generate project IDs** business rule, an ID appears automatically if you do not provide one.

Deleting Records Using Data Objects

Deleting a record is very similar to loading a data object, but it requires an additional line of code to instantiate the delete method.

The following code sample illustrates deleting a data object.

```
Dim oAcct as CGLAccount
Set oAcct = New CGLAccount

oAcct.Init FE_Application.SessionContext

'Load the first record
'Note: we left out some error trapping here (for example if this
'record didn't exist) to keep the sample clear
oAcct.Load 1

'Delete the Record using the Data Object's Delete method
oAcct.Delete

oAcct.CloseDown
Set oAcct = Nothing
```

Managing Child Objects

You can access a child object only via its parent object. In code, child objects require different procedures than top-level objects and are referred to through a left-to-right sequence that progresses from parents to child objects. For example, in the sample below, the top-level object oAccount is listed first, followed by the child object Notepads, followed by the notepad property IsMember, followed by the type True:

```
oAccount.Notepads.IsMember = True
```

For an overview of child objects, see “Understanding Child Objects” on page 10.

Adding a Child Object

To create a new child object, you must first create a top-level object with a child collection to contain the child object. Every child object must have a parent object and must be contained within a child collection. It is important to remember that no changes are made to the database until you call the parent record’s Save method. Next, use the collection’s Add method to return a new child object. At this point, the object is a member of the collection, but it is not added to the database until you call the Save method.

The following code sample illustrates adding a notepad to a *General Ledger* project:

```
Dim oProject As CGLProject
Dim oNotepad As IBBNotepad
'Example of Interface, this is not an object

'Create and load top object
Set oProject = New CGLProject
oProject..Init FE_Application.SessionContext
oProject.Load 9

'Create the notepad object.
Set oNotepad = oProject.Notepads.Add

oNotepad.Fields(NOTEPAD_fld_NotepadType) = "Internal"
'Notepad Type is the only required field.
oNotepad.Fields(NOTEPAD_fld_Author) = "George"
oNotepad.Fields(NOTEPAD_fld_Title) = "Example"
oNotepad.Fields(NOTEPAD_fld_Description) = "Example of child object"
oNotepad.Fields(NOTEPAD_fld_NotepadDate) = "05/15/2005"

'Child object is saved with top object.
oProject.Save

'Clean up child object, child is closed with top object.
Set oNotepad = Nothing

'Clean up top object.
oProject.CloseDown
Set oProject = Nothing
```

The collection’s Add method is the only way to create a new child object. All child objects are added using the same process.

Deleting a Child Object

Deleting a child object is very similar to adding a child object. First you must load the parent object, then you call the collection’s Remove method. This removes the child object from the collection, but you must call the parent’s Save method before the object is actually removed. Similar to the Item method, the Remove method is “overloaded”, providing two different ways to specify the child object to remove. The Remove method accepts either the actual object or the object’s ID number as parameters. It is important to remember changes are not actually made to the database until the parent record’s Save method is called.

The following code sample illustrates deleting a transaction from an existing batch record:

```
'This removes oTrans from the collection.
oBatch.Transactions.Remove oTrans

'This removes the 2nd element from the collection.
oBatch.Transactions.Remove 2

'The object is not actually removed from the database
'until this step.
oBatch.Save
```

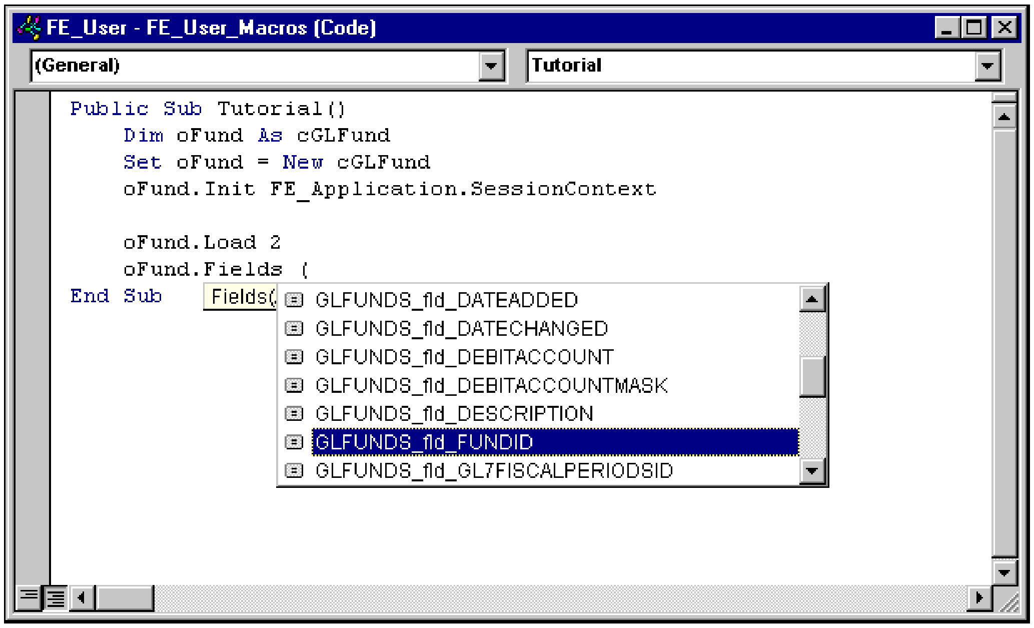
Whether you delete objects by the object name or ID, you get the same results. The situation determines the best method to use.

No warning message appears when deleting child objects, so you should add a warning that the user is about to delete important information.

Updating Data Objects

Each data object shares a common and very important property — Fields. Instead of exposing a unique property for each field on a data object, Blackbaud's developers built the Fields property to expose all the individual updatable data elements that make up a data object. This approach is much less cumbersome and is easier to extend. With the Fields property, when you access the Fields property from code, a list appears showing the constants for all valid fields on the object. This way, there is no time spent searching through hundreds of properties on an object just to find, for example, the "Full Name" field. This design also enables Blackbaud developers to easily add new fields as the accounting product evolves, without breaking any existing code.

The following code sample shows a list of constants for fund object fields.



```
Public Sub Tutorial()
    Dim oFund As cGLFund
    Set oFund = New cGLFund
    oFund.Init FE_Application.SessionContext

    oFund.Load 2
    oFund.Fields (
End Sub
```

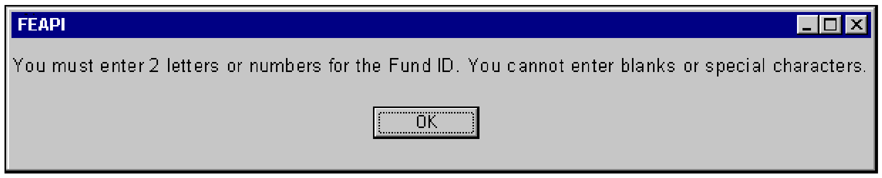
```
GLFUNDS_fld_DATEADDED
GLFUNDS_fld_DATECHANGED
GLFUNDS_fld_DEBITACCOUNT
GLFUNDS_fld_DEBITACCOUNTMASK
GLFUNDS_fld_DESCRIPTION
GLFUNDS_fld_FUNDID
GLFUNDS_fld_GL7FISCALPERIODSID
```

The following code sample illustrates loading a fund from the database into a fund data object, changing the fund ID, and saving the fund. If a user enters invalid data into the fund ID field (for example, “xxxx”) when the Save method is called, the data object raises a trappable error.

```
Dim oFund As CGLFund
Set oFund = New CGLFund

oFund.Init FE_Application.SessionContext
oFund.Load 2

'Change the fund description field
oFund.Fields(GLFUNDS_fld_FUNDID) = "25"

'Save changes
oFund.Save

'Clean Up
oFund.CloseDown
Set oFund = Nothing
```

Validating Data Objects

Validating data objects is much more complex than simply filtering out bad data. When validating, *Visual Basic* checks every business rule in the program, regardless of whether they are custom internal rules or rules you established in *Configuration*. For example, if end users attempt to enter a three-digit Fund ID when the program is configured for two-digit funds, an error message appears:



You must enter 2 letters or numbers for the Fund ID. You cannot enter blanks or special characters.

If you attempt to enter an invalid fund ID using *Visual Basic* code, a trappable error is raised with the same message (accessible via the Err.Description property on the *Visual Basic* Error object). This validation exists to maintain a high level of consistency and integrity in your database. The object insulates your database and prevents “garbage” from entering the database by first validating it. This rule applies to every facet of the data element, so you can be sure that updates using data objects are consistent with updates made by end users in the program.

Handling Data Object Errors

Before you resolve errors generated during program processing, it is important to understand the possible ways objects communicate with your programming. As you program, many times objects need to return information to the programs. For example, if you tried to use an account query to filter a project collection, obviously the query would not produce acceptable results because an account query cannot find projects. In this case, an error occurs. The query object needs some way to communicate this back to the program so you can realize a problem exists and then fix it. You can use two methods to report errors:

- You can use return values. With return values, object methods return an error code if your code contains errors. One advantage of this is that it enables you (in fact, it almost forces you) to handle every possible error as it happens. However, it can be cumbersome to explicitly check for every possible error in your code.

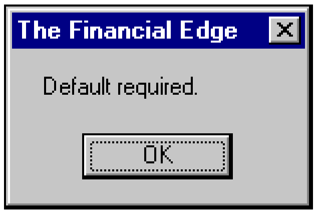
- You can also use *Visual Basic's* built-in capability to raise errors. This is the method used by our accounting objects. If proper error handling is not in place, these errors can cause the program to abort. Fortunately, handling errors in *Visual Basic* is very simple and offers many flexible ways to deal with errors. Depending on how you structure error handlers, you can handle each error in the subroutine in which it occurs, allow it to cascade back to a central error handler for the entire program, or use a variation of the two.

Using the Err.Description Property

When an error occurs, you can access information about the error by using the Err object provided by *Visual Basic*. Err.Description is a helpful property that tells you the reason for the error, such as failing to specify all required fields when adding a new *General Ledger* fund through code.

```
    Dim oFund As CGLFund
    Set oFund = New CGLFund

    oFund.Init FE_Application.SessionContext
    oFund.Fields(GLFUNDS_fld_DESCRIPTION) = "Operating Fund"

    On Error GoTo ErrorHandler
    oFund.Save

    'This turns off the Error Handler.
    On Error GoTo 0

    'Clean up!
    oFund.CloseDown
    Set oFund = Nothing

'Always place an Exit Sub before the Error Handler
'to prevent entering the Error Handler unintentionally.
Exit Sub

ErrorHandler:

    MsgBox Err.Description, vbOKOnly

    'This returns processing back to the line after where the error occurred
    Resume Next

End Sub
```

If we run this code, the CGLFund object raises an error because not all required fields are defined. An error message appears. (The message may vary depending on your configuration):


The Financial Edge

Default required.

Using the SessionContext ErrorObject

The error object accessed via the SessionContext also provides rich error details that are set by the last data object to raise an error (err.number=bbErr_DataObjectError). For more information, see the IBBErrorObject in the Object Reference.

The following table displays the properties and methods for the IBBErrorObject:

Property or Method	Type	Description
ErrorDescription	Get/Let	Set or get the description of the error.
ErrorNumber	Get/Let	Set or get the error number.
InvalidField	Get/Let	Set or get the invalid field number.
InvalidGenericObject	Get/Set	Set or get a reference to a generic object. (Used when InvalidObject is nothing)
InvalidMetaObject	Get/Set	Set or get a reference to the invalid IBBMetaField.
InvalidObject	Get/Set	Set or get a reference to the invalid IBBDataObject.
InvalidRule	Get/Let	Set or get the invalid rule number.
Clear	Sub	Clear any prior error values.
RaiseGenericError	Sub	Raise the specified error.
TranslateError	Function	Get the plain language translation of the specified error number.

Handling Warning Rules

Some objects have warning rules that need to be handled during a save. For instance, you might have a vendor record with a credit limit of $100 and try to save a new invoice of $200 for this vendor. Upon save, the following error is raised: "This invoice may cause you to exceed the current credit limit of $100.00 for this vendor". This is a warning rule, in that it can be overridden. Example:

```
Private Function AddInvoice()

    Dim oInvoice As CAPInvoice
    Set oInvoice = New CAPInvoice

    With oInvoice
        .Init gosessioncontext
        .Fields(APINVOICES_fld_AP7VENDORSID) = lVendorID
        .ApplyVendorDefaults True
        .Fields(APINVOICES_fld_INVOICENUMBER) = lInvoiceNumber
        .Fields(APINVOICES_fld_DESCRIPTION) = sInvoiceDescription
        .Fields(APINVOICES_fld_DUEDATE) = dtInvoiceDueDate
        .Fields(APINVOICES_fld_INVOICEAMOUNT) = CurInvoiceAmount
        .Fields(APINVOICES_fld_INVOICEDATE) = dtInvoiceDate

        SaveInvoice oInvoice
        .CloseDown
    End With

    Set oInvoice = Nothing

Exit Function

Private Function SaveInvoice(oInvoice As CAPInvoice) As Boolean

    Dim bResumeSave As Boolean
    Dim lErrNum As Long
    Dim sErrMsg As String

    On Error GoTo ErrHandler

    oInvoice.Save

Exit Function

ErrHandler:

    ' always cache error information before continuing
    lErrNum = gosessioncontext.ErrorObject.ErrorNumber
    sErrMsg = gosessioncontext.ErrorObject.ErrorDescription
    HandleError bResumeSave, lErrNum, sErrMsg
    If bResumeSave Then
        bResumeSave = False
        SaveInvoice = SaveInvoice(oInvoice)
    End If
End Function
```

(Continued- page 2 of 2)

```
Private Sub HandleError(ByRef bResumeSave As Boolean, ByVal lErrNum As Long, _
                          sErrMsg As String)

    If lErrNum = bbErr_OBJECT_WARNING Then
        Select Case gosessioncontext.ErrorObject.InvalidRule
          Case APInvoice_Warning_ExceedsVendorCreditLimit
            'in this example we'll just override the rule, but you could prompt the user.
            Dim oWarningRule As IBBWarningRule
            Set oWarningRule = gosessioncontext.ErrorObject.InvalidObject
            oWarningRule.OverrideWarning(APInvoice_Warning_ExceedsVendorCreditLimit)= True
            Set oWarningRule = Nothing
            'try to save again
            bResumeSave = True
          Case Else
            'test for other warning rules here...
            MsgBox sErrMsg
        End Select

    Else

        MsgBox sErrMsg

    End If

End Sub
```

Error handling is a very important part of programming. If you carefully trap and handle errors in *Visual Basic*, objects automatically communicate back to your program when they encounter errors, so programming becomes much simpler.

Managing Data Collections

Data collections are simply groups of data objects that share common characteristics. Following standard collection object model design practices, the accounting package always has two closely related classes that handle exposing collections: the parent, which is always named in the plural form (for example, Actions), and the child, which is always named in the singular form (for example, CAction).

Managing Top-Level Collections

Each top-level object has a corresponding top view collection. A distinguishing characteristic of a collection is that the object’s name takes the plural form. For example, the top view collection object for CGLAccount is CGLAccounts. For an introduction to top-level collections, see “Understanding Top-Level Collections” on page 12.

Managing Child Collections

Not all child objects are exactly the same. The various types of child objects/collections and the mechanics of programming objects and collections differ:

- Collection Type 1 — The Standard Child Collection

- Collection Type 2 — The Child View Collection

The most common use of child objects in the ***The Financial Edge*** object model is via standard child collections. A child collection, which is a collection of child objects, cannot exist without a top-level object. You can add and remove Child objects from the collection, but you cannot save Child objects without calling the parent's Save method. Attributes, notes, and the history of changes are popular examples of child objects that exist on numerous record types. Child objects cannot be directly created, loaded, saved, initialized, or deleted. All these actions are accomplished via methods exposed by the child object's parent object.

Child objects depend on the parent's Save method because the parent may have to enforce rules governing membership in the collection. When you save the parent, you also save all the child objects in the collection if they are dirty (a change has been made). Also, all objects you remove from the collection are deleted from the database. For more information about managing child view collections, see "Managing Child View Collections" on page 37.

Child objects depend on the parent's Save method because the parent may have to enforce rules governing membership in the collection.

Accessing Specific Child Elements

Like any *Visual Basic* collection, you can access ***The Financial Edge*** child objects directly via the item method. Things get a little tricky here, though. Because a child collection provides high-level access to underlying database records, Blackbaud's developers needed to "overload" the behavior of the item method and provide multiple ways to use it depending on the context from which you access the item. For example, if you pass in a string such as "5", the item method returns the child object with a database ID of "5". If you pass in a number such as 5, the item method returns the 5th member of the collection.

These two methods exist to ensure consistent access to collections across the object model. The most common use of the item method of a child collection is to pass it a numeric parameter, accessing the "nth" item. The value of accessing collection elements via their database ID will become more clear when you begin working with top-level collections.

The following code sample shows using the overloaded Item method to return elements.

```
'Access the 5th element in the collection
With oProject.Contacts.Item(5)
    Debug.Print .Fields(GLPROJECTCONTACTS_fld_ORGANIZATION)
End With

'Access an element of the collection that has an
'underlying database ID (primary key) of 5.
With oProject.Contacts.Item("5")
    Debug.Print .Fields(GLPROJECTCONTACTS_fld_ORGANIZATION)
End With
```

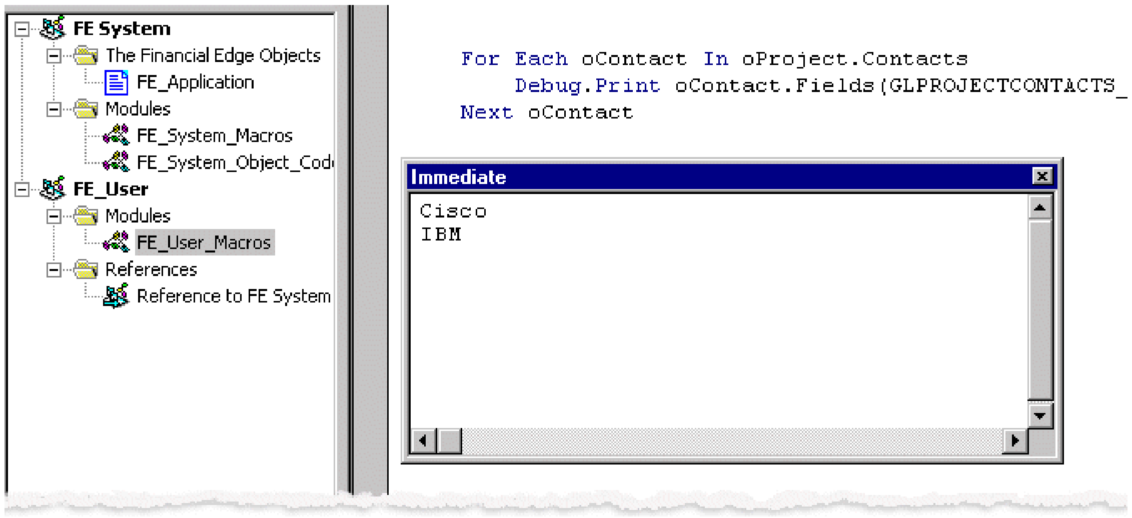
Iterating through Child Object Collections

The easiest, most efficient manner for iterating (or *navigating*) through child collections is through the *Visual Basic* "For Each" syntax. All collections support this format. An example of iterating is a code snippet in which you print a list of each contact attached to a project. When the last code is accessed, the loop automatically terminates. This process is infinitely faster, both in terms of coding and executing, than listing individual child objects in the collection.

The following sample code illustrates printing a list of project contacts from a collection using “For Each”.

```
'Code to initialize and load a CGLProject object (oProject)
'omitted for brevity.
Dim oContact as CProjectContact

For Each oContact In oProject.Contacts
    Debug.Print oContact.Fields _
      (GLPROJECTCONTACTS_fld_ORGANIZATION)
Next oContact
```

The following picture displays the results of the code sample.



```
For Each oContact In oProject.Contacts
    Debug.Print oContact.Fields(GLPROJECTCONTACTS_
Next oContact
```

```
Cisco
IBM
```

Updating Child Collection Elements

After you learn to access members of a child collection, you are only a few steps from learning to globally change them. To modify a group of child objects, you add a line of code that updates the child data via its Fields property. For example, to change the organization name of each contact, you add a new line of code setting the oContact.Fields property to the new value. It is very important to note that changes are not immediately committed to the database. Child objects do not have a Save method; the top-level parent is responsible for the save. When you save the top-level object, all changes are validated against any program and end user business rules. If the changes “pass” all the rule checks, the changed records are committed to the database. If a rule cannot be validated, ***The Financial Edge*** raises a trappable run-time error. Error checking is critical for preserving database integrity. The same rules that apply to an end user apply to your data objects. For more information about error trapping, see “Handling Data Object Errors” on page 30.

The following code sample illustrates globally changing the organization name of all contacts to XYZ, Inc.:

```
'Code to initialize and load a CGLProject object (oProject)
'omitted for brevity.
Dim oContact as CGLProjectContact

For Each oContact In oProject.Contacts
    Debug.Print oContact.Fields(GLPROJECTCONTACTS_fld_ORGANIZATION)

    'Modify each Contact, changing the organization name
    oContact.Fields(GLPROJECTCONTACTS_fld_ORGANIZATION) = "XYZ, Inc."
Next oContact

'Important! None of the changes are saved until
'the next line of code executes
oProject.Save
```

Managing Child View Collections

Child view collections are collections in which you can navigate to a subset of a particular true child collection. You cannot add to or remove from these collections using standard collection methods because they are just views of another collection and their membership is determined by other factors, such as very specific methods on the parent object. A good example of a child view collection is the Changes (History of Changes) collection exposed by the Account object. When you edit a property of the account, information about the change is stored in the Changes collection. With the Changes collection, you can view these changes. For a list of child collection methods, see “Understanding Child Collections” on page 13.

Sorting Collections

After you know how to access and move through collections, you may want to arrange objects in a different order from the way they normally appear in the collection. Not all collections can be sorted in this way, but many of the more commonly used collections support sorting.

When sorting collections, there are a couple of important things to keep in mind. First, remember when using the Item method, it returns the “nth” member based on the current sort. Second, when using top view collections, it is possible to filter out top-level objects using a query.

If you filter the collection using a query, the query order is retained regardless of the settings. You can sort using either of two properties. For more information about filtering collections with a query, see “Filtering Collections” on page 38.

SortField

Use the SortField property to designate any data field available in the member object as the sort field for the entire collection. With IntelliSense and Enums, it is very easy to select the field to sort by.

SortOrder

With the SortOrder property, you can sort in either ascending or descending order. If you do not specify a SortOrder, the default order is ascending.

The following code sample lists account descriptions in descending order:

```
'Initialize collection.
Dim oAccounts As CGLAccounts
Set oAccounts = New CGLAccounts
oAccounts.Init FE_Application.SessionContext

'Set the Field and Order for the sort.
oAccounts.SortOrder = Descending
oAccounts.SortField = GLACCOUNTS_fld_DESCRIPTION

'Declare an instance of the top object.
Dim oAccount as CGLAccount

'Loop through the collection.
For Each oAccount In oAccounts
    Debug.Print oAccount.Fields(GLACCOUNTS_fld_DESCRIPTION)
Next oAccount
```

Filtering Collections

Collections contain many methods and properties that make it easy to move through them to gather information. If you do not need to see all the child objects in a collection, you can use a query to filter the child objects. In top-level collections, you can filter child objects based on a query.

To filter a top-level collection, use the ID of the query you want to use to filter. If you know the name of the query, you can get the query ID by using the LoadByField method explained in “Loading Top-Level Data Objects” on page 23. Otherwise, you can iterate through the QueryObjects collection to find the query. The query type must match the record type of the collection. For example, if you use an oAccounts collection, the query you use must be an account query. If you specify a query of the wrong type, a trappable error message appears.

Once you know the query ID, you set the property FilterQueryID equal to this query ID. The collection returns only child objects contained in that query. Note that child objects are sorted into the collection in the same order as in the query.

The following code sample shows filtering accounts using the “Expenses” query:

```
Dim oQuery As CQueryObject
Set oQuery = New CQueryObject

oQuery.Init FE_Application.SessionContext

'This loads the query that is named Expenses.
oQuery.LoadByField uf_QUERY_NAME, "Expenses"

Dim oAccounts As CGLAccounts
Set oAccounts = New CGLAccounts

oAccounts.Init FE_Application.SessionContext

'This tells the collection which query (Expenses) to filter with.
oAccounts.FilterQueryID oQuery.Fields(QUERIES7_fld_QUERIESID)

'From here on, we can use the oAccounts collection and it will only
'contain CGLAccount objects that are in the Expenses query.
```

Managing User Interface Objects

To get the information you need from the end user, you must have some sort of user interface (UI). In many instances you must design a custom form to accomplish your program’s goals, but in some cases, you can use a form that already exists in ***The Financial Edge***. Using an existing form in your project is a simple way to save programming time, and it makes the program easier for the end user because they are already familiar with the forms. In ***The Financial Edge***, all top-level objects have user interface forms. When a form appears, it is fully functional and contains all toolbars and menus, so end users can perform the same operations they normally do within the program.

You can use many user interface forms in the program. Although different forms may have some different methods and properties depending on their use, all forms do have some things in common:

- They always have an Init method that accepts a SessionContext and a CloseDown method.
- They have a property that accepts a data object that “matches” the form.
- They always have the ShowForm method. This is what actually displays the form. ShowForm accepts four optional parameters:

Parameter	Variable Type	Description
bModal	Boolean	Determines whether the form appears modally. Defaults to False.
oFormToCenterOn	Object	The UI Form appears centered over the form specified here.
bDoNotCloseDataObject	Boolean	If set to True, the data object passed to the form is still initialized and may be used after the user closes the form.
oMoveServer	IBBMoveServer	This establishes how the “VCR” buttons on the form function.

An example of a user interface form is the CGLAccount. The CGLAccountForm needs an Account object so it can make changes or create a new record based on user interaction. If a new data object is passed and the user saves the form, the program creates a new record. If an existing data object is passed and the user saves the form, the program saves changes to the existing record. You can also use the data object’s fields property to fill some fields before the user views the form.

The following code sample illustrates displaying a user interface form to create accounts:

```
Dim oAccount as CGLAccount
Set oAccount = New CGLAccount
oAccount.Init FE_Application.SessionContext

'If we wanted to show an existing Account, we would load it here.
'Or we could set some of the .Fields before we display the Account.
oAccount.Fields(GLACCOUNTS_fld_ACCOUNTNUMBER) = "06-1111400-00-00"
oAccount.Fields(GLACCOUNTS_fld_DESCRIPTION) = "Investment Asset"
oAccount.Fields(GLACCOUNTS_fld_STATUS) = "Active"
oAccount.Fields(GLACCOUNTS_fld_CASHFLOW) = "Cash and Cash Equivalents"
oAccount.Fields(GLACCOUNTS_fld_WORKINGCAPITAL) = "Current Assets"

Dim oForm as CGLAccountForm
Set oForm = New CGLAccountForm
oForm.Init FE_Application.SessionContext

'This must be done first or an error is raised.
Set oForm.AccountObject = oAccount

'This displays the form modally, centered over frm_Main.
oForm.ShowForm True, frm_Main

'Clean up!
oForm.CloseDown
Set oForm = Nothing

oAccount.CloseDown
Set oAccount = Nothing
```

Visual Basic Interfaces

One of the powerful features of *Visual Basic* is that it supports the use of interfaces. An interface is an advanced programming technique ***Financial Edge*** developers use to make the program code more efficient and easier to maintain. An interface is like a contract. Any class that implements a specific interface guarantees the class supports a certain type of behavior. Many objects in ***The Financial Edge*** object model implement other interfaces. For example, all data objects, such as CGLFund, or CGLAccount, implement the IBBDataObject interface. This provides a way for you to refer to any data object that implements IBBDataObject in a generic fashion. Referring to the IBBDataObject interface gives you access to some properties and methods not available when referring to the actual class.

Using the IBBDataObject Interface

By referring directly to the IBBDataObject interface, you can use the Initialized property to find out whether an object has been initialized, and you can use the Dirty property to determine whether an object has changed since it was last saved.

Another important property of the IBBDataObject interface is the ObjectName property. This returns the name of the class of object you are using. One of the most important properties is the MetaField property, which provides access to information about the types of data a field expects. In the following code sample, notice that it includes a reference to the IBBTopObject interface. All top-level objects implement this interface. Using this interface provides a way to refer to and access the methods and properties common to all top-level objects.

```
Private Sub UsingInterfaces(oTopObject As IBBTopObject)

    Dim oIBBDataObject As IBBDataObject
    Set oIBBDataObject = oTopObject

    'This makes sure that the object passed in has been initialized.
    If Not oIBBDataObject.Initialized Then
        oTopObject.Init FE_Application.SessionContext
    End If

    'Allows you to find out what class has been passed
    Select Case oIBBDataObject.OBJECTNAME
        Case "CGLAccount"
            oIBBDataObject.Fields(GLACCOUNTS_fld_DESCRIPTION) = "New Account"
        Case "CGLAccountCode"
            oIBBDataObject.Fields(GLACCOUNTCODES_fld_DESCRIPTION) = "New Account Code"
        Case "CGLProject"
            oIBBDataObject.Fields(GLPROJECTS_fld_DESCRIPTION) = "New Project"
    End Select

    If oIBBDataObject.Dirty Then oTopObject.Save

    'Clean Up!
    Set oIBBDataObject = Nothing

End Sub
```

Using the IBBMetaField Interface

The IBBMetaField interface provides a convenient way to look up and change information about individual data fields used in ***The Financial Edge***. For example, in designing your programs, you may need to know if a particular field has been defined as required in *Configuration*, or you may need to know the type of data a specific field requires, such as date, number, or percentage.

Every data object in ***The Financial Edge*** object model provides an IBBMetaField interface to determine this information. Each data object has a Fields property and an array of the actual data in each field. Most of the properties in the IBBMetaField interface return a similarly numbered array. This makes using the IBBMetaData and the actual data object together much simpler.

Property	Return Value	Description
DisplayText	String	This returns the user-defined Display As text for this field.
FormatDescription	EFormatDescriptors	This returns the type of data contained in this field.
Required	Boolean	This returns True if the field is required.
UserRequired	Boolean	This returns True if this is a field the user has selected to make required.
UserHidden	Boolean	This returns True if the user has selected to make this field hidden from view.

You can set some properties through your program. For example, you can change DisplayText, UserHidden, and UserRequired. The Save method saves those changes to the database.

The following code sample shows using these IBBMetaField properties to load an array of text boxes and accompanying labels and enable changing the Display Text. The IBBDataObject interface returns a reference to the IBBMetaField interface.

```
Option Explicit

Private moAction As CGLAccount
Private moMetaField As IBBMetaField
Private moDataObject as IBBDataObject

Private Sub Form_Load()

    Dim i As Integer

    Set moAction = New CGLAccount
    moAction.Init FE_Application.SessionContext

    'Provides access to the IBBMetaField interface for the moAction object
    Set moDataObject = moAction
    Set moMetaField = moDataObject.MetaField

    For i = 1 to moMetaField.Count

        'If the field should be hidden we want to skip it.
        If Not moMetaField.UserHidden(i) Then
            'This is determined in Configuration.
            Label1(i).Caption = moMetaField.DisplayText(i)

            'You need to check if it is system-required or
            'if it is user-required.
            If moMetaField.Required(i) Or moMetaField.UserRequired(i) Then
                Label1(i).ForeColor = vbRed
            End If
            If moMetaField.FormatDescriptor(i) = fmtAMOUNT Then
                Text1(i).Text = "$" & Text1(i).Text
            End If

        End If
    Next i

End Sub
Private Sub Label1_DblClick(Index As Integer)

    Dim s As String
    s = InputBox("Enter the new Display Text")

    If Len(s) Then
        moMetaField.DisplayText(Index) = s
        'This must be called to save the user's changes.
        moMetaField.Save
    End If
```

(Continued- page 2 of 2)

```
End Sub

Private Sub Form_Unload(Cancel As Integer)

    'Clean up!
    Set moMetaField = Nothing
    set moDataObject = Nothing

    moAction.CloseDown
    Set moAction = Nothing

End Sub
```

Managing Service Objects

Using service objects, you can access discrete functionality within the application. Service objects include, but are not limited to, queries, reports, viewers, search screens, and some forms. A major advantage of service objects is that they enable you to quickly assemble solutions that leverage existing ***Financial Edge*** functionality, while at the same time presenting users with a familiar interface.

Managing Query Objects

A query is composed of a group of objects that provides query functionality in ***The Financial Edge*** object model. These objects include:

- CQueryObject
- CQueryObjects
- CQuerySet
- CStaticQ

These objects provide programmatic access to existing queries, provide access to the output of a query, and enable you to create a new static query you can use elsewhere in ***The Financial Edge***.

Opening a Query

Opening an existing query works like opening a data object. You access information about a query through the CQueryObject. First, you must initialize the object and then load it. Like data objects, you can use a Load method if you know the query ID. You can also loop through the CQueryObjects collection to locate the query. After you load the query, you can access its result set.

The following code sample displays loading a query with a database ID of 5:

```
Dim oQuery as CQueryObject
Set oQuery = New CQueryObject

oQuery.Init FE_Application.SessionContext

'Load the query using the Database ID
oQuery.Load 5
```

Processing a Query Result Set

By processing a query result set, you can move line by line through the results of a query. You can access a query result set in two ways. If you are already using a CQueryObject, you can access its result set with the Queryset method:

```
Dim oQuery as CQueryObject
Set oQuery = New CQueryObject

oQuery.Init FE_Application.SessionContext

'Load using the query name
oQuery.LoadByField uf_QUERY_NAME,"Account Managers"

'This opens the result set for access
oQuery.QuerySet.OpenQuerySet
```

Or, if you know the query's database ID, you can start with a CQueryset object:

```
Dim oQuery as CQuerySet
Set oQuery = New CQuerySet

oQuery.Init FE_Application.SessionContext

'This uses the database ID of the query
oQuery.QueryID = 10

'This opens the result set for access
oQuery.OpenQuerySet
```

Both of these examples accomplish the same thing. In either case, you must reference a query result set. You can use a few properties to help access the data from the result set:

Property	Returns
FieldCount	The number of fields in the output of the query
FieldName	An array of the field names in the output
FieldType	An array of the field types, for example, Date, Double, Long, Memo, Text
FieldValue	An array of the actual data for the current row
RowNum	The number of the current row

At this point, you can loop through the result set to gather the data you need:

```
Debug.Print oQuery.FieldName(1) & " " & oQuery.FieldName(2)

Do While Not oQuery.EOF
    'This is where you would access the fields.
    Debug.Print oQuery.FieldValue(1) & " " & oQuery.FieldValue(2)
    oQuery.MoveNext
Loop

'Clean up.
oQuery.CloseDown
Set oQuery = Nothing
```

Creating Static Queries

Using the CStaticQ object, you can create static queries in your code. Static queries are lists of unique IDs. If you create a static query in your code, you cannot open it in *Query* because it has no sort, filter, or output fields. However, other queries and processes that use queries, such as *Mail* or *Reports*, can use static queries. Static queries are ordered and have no duplicates. To create a static query, you use three methods plus the Init and CloseDown methods.

➢ Creating a static query using the CStaticQ object

1. To create a new query, use the Create method, which displays the same Create Query form used in ***The Financial Edge***. The user specifies the name and other information about this query. If the user clicks **Cancel**, the Create method returns False. Abort the process if the Create method returns False. For information about aborting the creation of a static query, see the EndCreate parameter **bCancel** in step 3.

Parameter	Variable Type	Description
SearchType	bbSearchTypes	Determines types of records included in the query
aFromProcessName	String	Each query stores the name of the area of the program in which it was created; you may put the name of your application here
FormToCenterOn	Object	The Create Query form displays itself centered over the object specified here
sDescription	String	Optional: you can input a default Description for the new query
lSystemID	Long	Optional
sDefaultQName	String	Optional: you can input a default name for the new query

2. To add the database IDs of records to include in the query, use the AddRecord method and pass the ID as the only parameter. The AddRecord method verifies this is not a duplicate ID and then adds it to the query. This is the only step required to add a record to the query.

3. To finish creating the query and write the information to the database, call the EndCreate method. Until this is called, the IDs are just stored in memory. EndCreate has three parameters:

   - **FormToCenterOn** accepts an object. When EndCreate is called, it normally displays a Writing Static Records form while it is writing IDs to the database. You can specify the form on which the Writing Static Records form centers itself.

   - **bCancel** is an optional parameter that defaults to False if nothing is passed. If your code allows a user to cancel the creation of the query after calling the Create method, it is important to call the EndCreate method and pass True for this parameter. The query is not created, but this frees the memory used to keep track of the IDs for the query.

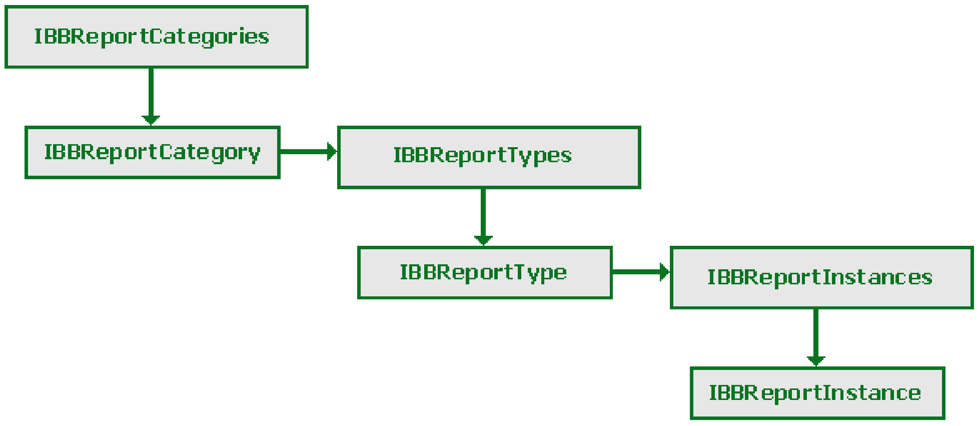
- **bNoUI** is an optional parameter. To keep the program from displaying the Writing Static Records form, set this to True.

```
Dim oProject As CGLProject
Set oProject = New CGLProject

Dim oProjects As CGLProjects
Set oProjects = New CGLProjects
oProjects.Init FE_Application.SessionContext

Dim oStaticQuery As CStaticQ
Set oStaticQuery = New CStaticQ

oStaticQuery.Init FE_Application.SessionContext

'This prompts the user for a Query Name but
'everything is already filled in.
If oStaticQuery.Create(SEARCH_GLPROJECT, "Custom App", Me, _
                        "Projects that have an Education type", , _
                        "Education Projects") Then

    For Each oProject in oProjects
        'This checks each project to see if its type is "Education".
        If oProject.Fields(GLPROJECTS_fld_TYPE) = "Education" Then

            'This adds the ID to the query.
            oStaticQuery.AddRecord
            oProject.Fields(GLPROJECTS_fld_GL7PROJECTSID)

        End If
        oProject.CloseDown
    Next oProject

    Set oProject = Nothing

    oProjects.CloseDown
    Set oProjects = Nothing

    'After we have all of our records in the query,
    'we write the data to the database.
    oStaticQuery.EndCreate Me, False, False

    oStaticQuery.CloseDown
    Set oStaticQuery = Nothing
Else
    'This means the user canceled when entering the query name.
    MsgBox "No query created", vbOKOnly
End If
```

Managing Report Objects

Report objects work together to access ***Financial Edge*** reports and mail functionality programmatically. Because mail functions also use *Crystal Reports*, it makes sense to provide one set of objects to print reports and process mail functions. These objects appear in a hierarchy that represents the way they are accessed in ***The Financial Edge***. For example, when you select *Reports* from the navigation bar, the Reports page appears, listing report categories such as **Financial Reports** or **Invoice Reports**. When you click a category, a list of the report types in that category appears, such as **Income Statement** or **Invoice History**. If you create a new report, a screen opens containing tabs on which you define parameters for that report. Report objects follow the same hierarchy, but, depending on the needs of your task, you can enter the object model from any object and create each class independently. You do not directly create report objects as you do other objects. To create a new ReportCategories or ReportCategory object, use the FE_Services object.

In this hierarchy, IBBReportCategories is a collection of IBBReportCategory objects. These represent the categories of reports or mail selections, such as Financial Reports or Invoice Reports in *Reports* or Forms in *Mail*. The next level is the IBBReportTypes and the IBBReportType objects — these represent specific reports such as the Balance Sheet, Income Statement, or Batch Detail Report. The last level of the hierarchy includes the IBBReportInstances and IBBReportInstance objects. These correspond to the individual parameter files you can save for each report. At this level, you can allow users to preview or print the report or create a new set of parameters for the report type.



Reports Categories Collection

Reporting and mailing functions are broken down by similar functionality into categories such as Financial Statements, Vendor Reports, and Forms. Each of these categories is represented by an IBBReportCategory object. Use the FEService object to create both IBBReportCategories and IBBReportCategory objects. For more information about these objects, see “Report Objects Example” on page 51. After you create an IBBReportCategories collection, use the Init method. This is similar to the Init method used for other objects to initialize them.

The following table lists parameters for Report Categories Collections' Init method:

Parameter	Variable Type	Description
SessionContext	IBBSessionContext	Used to initialize all objects.
ActiveSystem	Long	Determines reports to include in the collection. General Ledger and Accounts Payable are examples of ActiveSystems. For a complete list of available arguments, see the Object Browser and search for 'ActiveSystem'.
lSubCategoryOfCategoryID	Long	Optional: Reserved for future use.
CategoryFilter	EBBRep_ ReportCategoryFilters	Optional: Use to specify whether to include only Report, Mail, or both IBBReportCategory objects in the collection. This defaults to include only Report IBBReportCategory objects.
QueMode	Boolean	Optional: Establishes how errors are addressed when using the collection. If set to True, an error log file is created, but the program continues to process. If set to False (the default), a trappable error is raised.
ShowMembers_ BasedOnSecuritySettings	Boolean	Optional: Use this to include only IBBReportCategory objects in the collection that a user has the security rights to view. If set to False, the collection includes all objects, regardless of the user's security; however, users cannot run reports without security rights. This parameter defaults to True.
ShowCannedReportsOnly	Boolean	Optional: There are some IBBReportCategory objects that represent reports not accessed via *Reports*, such as control reports in *Query* or Global Add. If this is set to True (the default) only the IBBReportCategory objects representing categories found in *Reports* are included in the collection.

When you initialize IBBReportCategories, you can use a "For Each" construct to loop through it, or you can use the Item property to access the IBBReportCategory objects in the collection. If you enter the report hierarchy directly from an IBBReportCategory object, you need to call its Init method, first. Some Init parameters for the IBBReportCategory are similar to parameters for the IBBReportCategories, but they work slightly differently. The IBBReportCategory object contains a ReportTypes method that returns an IBBReportTypes collection of IBBReportType objects. The ShowMembersBasedOnSecuritySettings and ShowCannedReportsOnly parameters filter the IBBReportType objects included in the IBBReportCategory.ReportTypes collection.

The following table lists parameters for Report Category Objects' Init method:

Parameter	Variable Type	Description
SessionContext	IBBSessionContext	Used to initialize all objects.
CategoryID	EBBRep_ ReportCategories	This is an Enum of all the categories of reports in ***The Financial Edge***.
QueMode	Boolean	Optional: Establishes how errors are addressed when using the object. If set to True, an error log file is created, but the program continues to process. If set to False (the default), a trappable error is raised.
ShowMembersBased_ OnSecuritySettings	Boolean	Optional: Use this to include only the IBBReportCategory.ReportTypes collection objects a user has security rights to view. If set to False, the collection includes all objects, regardless of the user's security; however, users cannot run reports without security rights. This defaults to True.
ShowCannedReportsOnly	Boolean	Optional: Some IBBReportType objects represent reports not accessed via *Reports*. If set to True (the default), only the IBBReportType objects representing reports found in *Reports* are included in the IBBReportCategory.ReportTypes collection.

When you initialize the object, you can access its ReportTypes property to move farther down the hierarchy of report objects. For more information about creating and using these objects, see "Report Objects Example" on page 51. For more information about other properties and methods of these two objects, see the Programming Reference in the VBA and API help file.

Reports Types Collection

You can access collections and objects that represent the highest level of the reports hierarchy — report categories. First, use the FE_Services object to create either an IBBReportTypes collection or an IBBReportType object. For more information about creating these objects, see “Report Objects Example” on page 51. Next, call the Init method. The Init method has some parameters that can filter the IBBReportType objects you want to include in the collection.

The following table lists parameters for Report Types Collections’ Init method:

Parameter	Variable Type	Description
SessionContext	IBBSessionContext	Used to initialize all objects.
CategoryID	EBBRep_ReportCategories	This is an Enum of all report categories so that only ReportTypes that are a part of this category are included in the collection.
QueMode	Boolean	Optional: This establishes how errors are addressed when using the collection. If it is set to True, a log file is created containing any errors, but the program continues to process. If set to False (the default), a trappable error is raised.
ShowMembers_ BasedOnSecuritySettings	Boolean	Optional: This includes in the collection only IBBReportType objects the user has security rights to view. If set to False, the collection includes all objects, regardless of the user’s security; however, users cannot run reports without security rights. This defaults to True.
ShowCannedReportsOnly	Boolean	Optional: Some IBBReportType objects represent reports not accessed via *Reports*, such as control reports in *Query* or Global Add. If set to True (the default), only the IBBReportType objects representing ReportTypes found in *Reports* are included in the collection.

After initializing the IBBReportTypes collection, you can iterate through the collection or select an IBBReportType object by using the Item method. If you already know the type of report you want to access, you can enter the report hierarchy at the IBBReportType object. As always, call the Init method and provide the parameters to access the correct report.

The following table lists parameters for Report Type Objects’ Init method:

Parameter	Variable Type	Description
SessionContext	IBBSessionContext	This is the same SessionContext used to initialize all objects.
ReportTypeID	EBBRep_ReportTypes	This is an Enum of all Report types so you can specify the report to access.
ShowOnlyMyReports	Boolean	Optional: If this is set to True, the collection contains only IBBReportInstance objects that represent parameter files created by this user. If set to False (the default), all parameter files available to the user are represented in the collection.

When you initialize the IBBReportType object, you can access its read-only properties to get more information about this particular report. It also has a ReportInstances property, so you can access the last levels of the report hierarchy. Here, you can actually process a report.

Report Instances Collection

You can process reports only at the lowest level of the report object hierarchy. Using the IBBReportInstances and IBBReportInstance objects, you can access any parameter files that already exist, and you can permit users to create new parameter files using the same forms they use in ***The Financial Edge***.

The IBBReportInstances object is a collection that represents all the parameter files for a particular type of report. As with the report objects, you create it with the FE_Services object. For more information about these objects, see “Report Objects Example” on page 51. When you create the object, use the Init method to initialize it. When the you initialize the collection, you can use any standard process to iterate through the collection.

The following table lists parameters for Report Instances Collections’ Init method:

Parameter	Variable Type	Description
SessionContext	IBBSessionContext	This is the same SessionContext used to initialize all objects.
ReportTypesID	EBBRep_ReportTypes	This is an Enum of all the Report types. Only ReportInstances for the type specified here are included in the collection.
QueMode	Boolean	Optional: This establishes how errors are addressed when using the collection. If set to True, a log file is created containing any errors, but the program continues to process. If set to False (the default), a trappable error is raised.
ShowOnlyMyReports	Boolean	Optional: If this is set to True, the collection contains only IBBReportInstance objects that represent parameter files created by this user. If set to False (the default), all parameter files available to the user are represented in the collection.

If you use the IBBReportInstance object to enter the hierarchy, first call the Init method. The following table lists parameters for Report Instance Objects’ Init method:

Parameter	Variable Type	Description
SessionContext	IBBSessionContext	This is the same SessionContext used to initialize all objects.
QueMode	Boolean	Optional: This establishes how errors are addressed when using the collection. If set to True, a log file is created containing any errors, but the program continues to process. If set to False (the default), a trappable error is raised.

After you initialize an IBBReportInstance, you can either load an existing parameter file or create a new one. To load an existing IBBReportInstance, all you need to know is the ReportParameterID, which is the database ID of the parameter file. After you use the Load method, or if you are creating a new parameter file, call the Process method. The following table lists parameters for the Process method of IBBReportInstance:

Parameter	Variable Type	Description
Action	EBBRep_ProcessOptions	This is an Enum of the process options available.
ShowModal	Boolean	Optional: This determines whether the Process form (this varies, depending on the action) is displayed modally. This defaults to False.
FormToCenterOn	Object	Optional: This determines over which object the Process form appears. If nothing is passed, the form appears in the center of the screen.

The Process method supports a number of actions that are enumerated as EBBRep_ProcessOptions. These include EBBRep_ProcessOption_ShowParameterForm; this shows the parameter form for the particular report type you are using. If you have not called the Load method, a new parameter form is displayed that allows the user to complete the parameters and save and run the report from the parameter form. If you have called the Load method, the form appears with the parameters already displayed, enabling users to edit the parameters and run the report. To not display the parameters, you can use the other EBBRep_ProcessOptions to print, print preview, export, send as mail, or view the report layout. For more information about other properties and methods available, see the Programmer’s Reference section of the VBA and API help file.

It is important that when you finish using an IBBReportInstance you call the CloseDown method. Even though it may return False, indicating it cannot be closed at this time, it sets an internal flag and cleans everything up as soon as the user closes the report. For example, after you call the Process method with an action of Preview, you may call the CloseDown method. When the user closes the preview window or exits the application, the object releases the resources it was using. However, you should make sure you do not need to access any property or method from the object after calling CloseDown because once it is called, the object acts as if it is closed down even if the preview window or parameter form is still open.

Report Objects Example

The following code sample illustrates using Report objects to add all possible report categories, types, and instances to a treeview.

```
Option Explicit

Private FEService As FE_Services

Private Sub Form_Load()

    Dim oReportCategories As IBBReportCategories
    Dim oReportCategory As IBBReportCategory
    Dim oReportTypes As IBBReportTypes
    Dim oReportType As IBBReportType
    Dim oReportInstances As IBBReportInstances
    Dim oReportInstance As IBBReportInstance

    'This is the class that we use to create the Report objects
    Set FEService = New FE_Services
    FEService.Init FE_Application.SessionContext

    Set oReportCategories = FEService.CreateServiceObject(bbsoReportCategories)

    oReportCategories.Init FE_Application.SessionContext,1, ,_
                            bbrep_ReportCategoryFilter_Reports, False, True, True

    For Each oReportCategory In oReportCategories

        TreeView1.Nodes.Add , , oReportCategory.CategoryName, _
                    oReportCategory.CategoryName

        'You can also use oReportCategory.ReportTypes
        Set oReportTypes = FEService.CreateServiceObject(bbsoReportTypes)
        oReportTypes.Init FE_Application.SessionContext, _
                    oReportCategory.CategoryID, False, True, True

        For Each oReportType In oReportTypes
            TreeView1.Nodes.Add oReportCategory.CategoryName, _
                        tvwChild, "Type" & Str$(oReportType.ReportID), _
                        oReportType.ReportName

            'You can also use oReportTypes.ReportInstances
            Set oReportInstances = _
                        FEService.CreateServiceObject(bbsoReportInstances)
            oReportInstances.Init FE_Application.SessionContext, _
                                        oReportType.ReportID, False, False

            For Each oReportInstance In oReportInstances
                With oReportInstance
                    TreeView1.Nodes.Add "Type" & Str$(oReportType.ReportID), _
                        tvwChild, _
                        Str$(.Property(bbrep_Property_ReportParameterNamesID)), _
                        .Property(bbrep_Property_Name)
```

(Continued- page 2 of 2)

```
                    .CloseDown
                End With
            Next oReportInstance

        Next oReportType

    Next oReportCategory

End Sub
```

Use Report objects to display parameters for the report instances users select so they can change parameters. Users can print, print preview, or save changes in the Reports section of ***The Financial Edge***. When the parameter form appears, the user can perform any normal ***Financial Edge*** operation without any additional code.

```
Private Sub TreeView1_DblClick()

    Dim lKey As Long

    'This makes sure that they have chosen an
    'Instance and not a Type or Category
    If Left$(TreeView1.SelectedItem.Key, 8) = "Instance" Then

        Dim oReportInstance As IBBReportInstance

        'This uses the Key from the parent (the Report Type) to specify what Type
        'of report this is.

        lKey = Int(Mid$(TreeView1.SelectedItem.Parent.Key, 5))

        Set oReportInstance = FEService.CreateReportInstance(lKey)
        oReportInstance.Init FE_Application.SessionContext

        'This uses the Key from the Instance to Load the correct parameter file
        oReportInstance.Load Int(Mid$(TreeView1.SelectedItem.Key, 9))

        'This displays the Parameter form. At this point the user can do
        'anything available in The Financial Edge.
        oReportInstance.Process bbrep_ProcessOption_ShowParameterForm, False, Me
        'At this point, we no longer need to access oReportInstance so we
        'call CloseDown. It will not be able to close right away but will close as
        'soon as the user closes the parameter form or exits the app.
        oReportInstance.CloseDown

    End If

End Sub
```

Using the Code Tables Server

In ***The Financial Edge***, a code table is a list of acceptable values for a particular data field. A user must select from that list or, if he or she has the proper security rights, add a new entry to the list. Code tables are used extensively throughout the program. For example, on account records, in the **Class** field, you can select from a list of classes. This same code table appears throughout the program. Allowing a user to select an entry from a list of options simplifies data entry, minimizes typing, and helps maintain consistency in data entry. You can reduce the size of the database by storing the number that relates to the table entry rather than the actual text. The CCodeTablesServer object provides many methods that make using code tables in your code much easier.

First, use the FE_Services object to create an instance of the CCodeTablesServer. Next, call the Init method and provide the SessionContext; you can then use any of the object's methods and its collection of CCodeTable objects, which contain information about all code tables in the program. Because there may be many opportunities to use a CCodeTablesServer in the program, you may want to place this initialization code in the Form_Load. When you no longer need the object, call the CloseDown method. You can place this in the Form_Unload. The LoadCombo method in the CCodeTablesServer provides a simple way to load a *Visual Basic* combo box with the entries for a particular code table.

The following table lists parameters for the LoadCombo method of the CCodeTablesServer:

Parameter	Variable Type	Description
oCombo	Object	This is the combo box you want to load.
lTableNumber	ECodeTableNumbers	This is an Enum of all of code tables available in the program.
bUseShort	Boolean	Optional: Some code tables have both short and long descriptions. Normally, you use the long description, but if you need to use the short description, set this to True. False is the default.
bActiveOnly	Boolean	Optional: In ***The Financial Edge***, you can mark table entries Inactive if they are not likely to be used anymore. If set to True (the default), only entries that are not flagged as Inactive appear.
bClearCombo	Boolean	Optional: If set to True (the default), any entries in the combo box are cleared before the combo is loaded.

The following code sample illustrates using the LoadCombo method to add table entries:

```
Option Explicit

Private moFEService As FE_Services
Private moCodeTablesServer As CCodeTablesServer

Private Sub Form_Load()

    'This is the class that we use to create the service objects
    Set moFEService = New FE_Services
    moFEService.Init FE_Application.SessionContext

    'This creates an instance of the CodeTableServer
    Set moCodeTablesServer = FEService.CreateServiceObject(bbsoCodeTablesServer)
    moCodeTablesServer.Init FE_Application.SessionContext

    'This loads the combo with the entries from the GLClass table
    moCodeTablesServer.LoadCombo Combo1, ctnumGLClass, False, True, True

End Sub

Private Sub Form_Unload(Cancel As Integer)

    moCodeTablesServer.CloseDown
    set moCodeTablesServer = Nothing

End Sub
```

If you provide the database ID, you can use the GetTableEntryDescription method to get the table entry description and use the GetTableEntryID method to obtain table entry IDs.

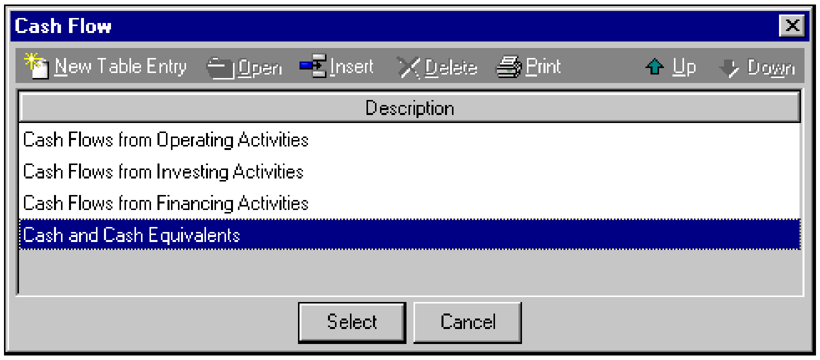
The following table illustrates retrieving the database ID and description from a *General Ledger* class table:

```
Dim lLong As Long
Dim sString As String

    'lLong will equal the database ID for the entry
    '"Unrestricted Net Assets" in the GLClass table. However this number
    'will vary from database to database.

    lLong = moCodeTablesServer.GetTableEntryID("Unrestricted Net Assets", _
                                              ctnumGLClass, False)

    'sString will equal "Unrestricted Net Assets"
    sString = moCodeTablesServer.GetTableEntryDescription(lLong, ctnumGLClass, False)
```

Using the Table Lookup Handler

The TableLookupHandler object works together with the CodeTablesServer to provide code table functionality present in ***The Financial Edge***. With the TableLookupHandler, you can add a new entry to a code table and display the table entry maintenance form used in the program so users can add, delete, or reorder code table entries. For more information about the CodeTablesServer, see “Using the Code Tables Server” on page 53.

The following picture displays the table entry maintenance form.



Description
Cash Flows from Operating Activities
Cash Flows from Investing Activities
Cash Flows from Financing Activities
Cash and Cash Equivalents

You use the FEService object's CreateServiceObject method to create an instance of the object and then call the Init method. Besides providing the usual SessionContext, you can also provide a reference to an existing CodeTablesServer object. This is not required, but it speeds the initialization process. As with the CodeTablesServer, it is best to place this in your Form_Load so you can use it throughout the form. You can place the CloseDown method in the Form_Unload to release all resources when you are finished. To display the maintenance form so a user can select, add, delete, and sort table entries, call the ShowForm method.

The following table shows parameters for the ShowForm method of CTableLookupHandler:

Parameter	Variable Type	Description
lCodeTableID	ECodeTableNumbers	This is the ID for the code table you want to appear.
lFindItemData	Long	Optional: This is the database ID for the table entry you want to have focus when the form appears.
oFormToCenterOn	Object	Optional: This is a reference to the form over which you want the maintenance form to appear. If nothing is passed, the form appears in the center of the screen.

Before you display this form, you can set two properties that influence how the form appears. If you set the ReadOnly property to True, the user will not be able to use the form to add, delete, or reorder table entries. If you set the ShowInactiveEntries property to True, table entries marked as Inactive are included on the form. The Canceled property returns a boolean that tells you whether the user canceled the form. The SelectedItem property returns the database ID of the table entry the user selects. If the user selects no item, it returns a 0 and if an error occurs, the property returns -1. In ***The Financial Edge***, if a user double-clicks the label for a table entry field, the maintenance form appears.

The following code sample creates a working code table maintenance form using the TableLookupHandler:

```
Option Explicit

Private moCodeTablesServer As CCodeTablesServer
Private moTableLookupHandler As CTableLookupHandler

Private Sub Form_Load()

    'Since the TableLookupHandler uses a CodeTablesServer object,
    'we can create it first.
    Set moCodeTablesServer = FEService.CreateServiceObject (bbsoCodeTablesServer)
    moCodeTablesServer.Init FE_Application.SessionContext

    Set moTableLookupHandler = FEService.CreateServiceObject(bbsoTableLookupServer)
    'We pass the reference to moCodeTablesServer to speed the Init process.
    moTableLookupHandler.Init FE_Application.SessionContext, moCodeTablesServer

End Sub

Private Sub Label1_DblClick()

    moTableLookupHandler.ReadOnly = True
    moTableLookupHandler.ShowInactiveEntries = True

    'By setting sFindItemData, if there is already a table entry in
    'the combo box, that entry will have focus when the form is displayed.
    moTableLookupHandler.ShowForm ctnumGLCashFlow, _
        moCodeTablesServer.GetTableEntryID(Combo1.Text, ctnumGLCashFlow), Me

    'If the user cancels the maintenance form then we don't want to change
    'what is already in the combo box.
    If Not moTableLookupHandler.Canceled Then
        'This uses the SelectedItem property to fill in the Combo box.
        Combo1.Text = moCodeTablesServer.GetTableEntryDescription _
          (moTableLookupHandler.SelectedItem, ctnumGLCashFlow, False)
    End If

End Sub
```

With the TableLookupHandler object, you can also add new table entries that do not have a short or long description to the table throughout the program by using the AddEntry method. When this method is called, the new entry is immediately added to the database.

The following table lists parameters for the AddEntry method of CTableLookupHandler:

Parameter	Variable Type	Description
bAddOnTheFly	Boolean	This should always be set to True so the new table entry is immediately added to the database.
lCodeTableID	Long	Optional: This is the code table number to which this table entry belongs. If you do not specify this, the current code table set within TableLookupHandler is used.
sShortDescription	String	Optional: This is the short description for this table entry.
sLongDescription	String	Optional: This is the long description for this table entry.
oForm	Object	Optional: The AddEntry method calls this object’s Refresh method.

The following code sample illustrates adding new table entries using the AddEntry method:

```
Private Sub Combo1_LostFocus()

    Dim sMsg as String

    If Len(Combo1.Text) > 0 Then

        With moCodeTablesServer

            'GetTableEntryID will return a 0 if the current text is not in
            'the table.
            If .GetTableEntryID(Combo1.Text, ctnumGLCashFlow, False) = 0 Then
                sMsg = "Do you want to add '" & Combo1.Text & "' to the " & _
                  .TABLENAME(ctnumGLCashFlow) & " table?"

                If MsgBox(sMsg, vbQuestion + vbYesNo) = vbYes Then
                        'This adds the current text to the database and
                        'Refreshes Combo1. If the AddEntry is unsucessful
                        'this will return False.
                    If Not moTableLookupHandler.AddEntry(True, _
                      ctnumGLCashFlow, , Combo1.Text, combo1) Then
                        MsgBox "Unable to add entry", vbInformation + vbOKOnly
                    End If


                Else
                    'If they don't want to add to the table, then they need to
                    'pick something that is already on the list.
                    Combo1.SetFocus

                End If

            End If

        End With

    End If

End Sub
```

Using the Attribute Type Server

The AttributeTypeServer object provides access to a collection of methods used to gather information about any of the attributes in ***The Financial Edge***. You can use this information to manage attributes on your custom forms. Attributes in the program consist of a category, description, date, and comment. When you create the attribute, the type of information contained in the description is also defined. The description can be in text, number, date, currency, yes/no, or table format. If the description type is a table, you may also want to use the CodeTablesServer and TableLookupHandler. For more information about the CodeTableServer, see “Using the Code Tables Server” on page 53. For more information about the TableLookupHandler, see “Using the Table Lookup Handler” on page 54.

First, use the FE_Services object to create a new instance of the AttributeTypeServer. After you create the object, call the Init method, passing a valid SessionContext. As with other service objects, we recommend you place this in the Form_Load so these methods are available throughout the form. You must also call the CloseDown method when you finish using the object, preferably in the Form_Unload.

The GetAttributeTypeID method requires two parameters, a String, which is the attribute category you are looking for, and an Enum of the different kinds of attributes (for example, account or project). The method returns a Long that is the database ID for this particular attribute. Once you know the attribute ID, you can use that ID to find out more information about the attribute. The inverse of this function is the GetAttributeTypeDescription. If you pass the attribute ID, it returns the attribute category as a String. Using the attribute ID, you can use the GetAttributeDataType method to find out the type of data required for the description of a particular attribute. This method returns a number that corresponds to a member of the bbAttributeTypes enum. The GetAttributeDataType method also accepts a boolean variable that is passed by reference, bUniqueRequirement. After the method is called, the variable is set to True if this attribute type allows only one attribute of this type per record.

If the data type for the attribute is a table, you may need to get the code table ID for the table. With this, you use the CodeTablesServer and TableLookupHandler to simplify your coding. When the GetAttributeCodeTableID method is passed to the attribute ID, it returns the code table ID for the table.

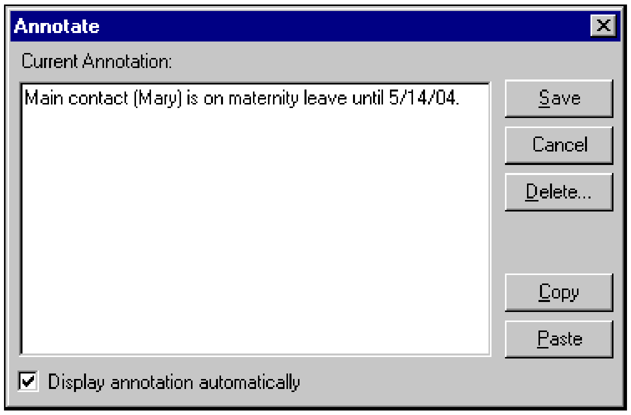
The following code sample displays an attribute category label and a combo box or text box, depending on the attribute type.

```
Option Explicit

Private moCodeTablesServer As CCodeTablesServer
Private moAttributeTypeServer As CAttributeTypeServer

Private Sub Form_Load()

    Dim lAttribute_ID As Long
    Dim bOnlyOneAllowed As Boolean

    Set moCodeTablesServer = moFEService.CreateServiceObject(bbsoCodeTablesServer)
    moCodeTablesServer.Init moFEAPI.SessionContext

    Set moAttributeTypeServer = moFEService.CreateServiceObject(bbsoAttributeTypeServer)
    moAttributeTypeServer.Init moFEAPI.SessionContext, bbGlobalAttributeType_GLAccount

    With moAttributeTypeServer

        lAttribute_ID = .GetAttributeTypeID("Budget Manager")
        Label1.Caption = .GetAttributeTypeDescription(lAttribute_ID)

        'bOnlyOneAllowed will now be True or False depending on if this
        'Attribute is allowed to be present more than once per record
        Select Case .GetAttributeDataType(lAttribute_ID, bOnlyOneAllowed)
            'If the Data Type is Boolean than we add Yes and No to the Combo box
            Case bbAttribute_BOOLEAN
                Combo1.Visible = True
                Combo1.AddItem "Yes"
                Combo1.AddItem "No"

            Case bbAttribute_TABLEENTRY
                Combo1.Visible = True
                'This uses the CodeTablesServer to the load the combo
                'with all of the table entries
                moCodeTablesServer.LoadCombo Combo1, _
                            .GetAttributeCodeTableID(lAttribute_ID), , True

            Case Else
                Text1.Visible = True

        End Select

    End With

End Sub
```

Using Annotation Forms

In ***The Financial Edge***, users can annotate any of the top-level data objects. An annotation is simply a note attached to each record. The user can also select to have this note appear whenever that record is loaded. By using the Annotation Form service object, you can easily add this functionality to your custom applications. Before you can use the Annotation Form object, you must declare an object reference for the FE_Services object.

The following picture shows an annotation form.



Main contact (Mary) is on maternity leave until 5/14/04.

☑ Display annotation automatically

➢ Using the Annotation Form object

1. Declare an object reference for the FE_Services object. Then create a new instance of the FE_Services object and use the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext to initialize it.
2. Declare an object reference for the CAnnotationForm object, then create a new instance of the CAnnotationForm object.
3. Set the CAnnotationForm object equal to the FE_Services.CreateServiceObject method, passing the Enum constant bbsoAnnotationForm.
4. Initialize the CAnnotationForm object using the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext.
5. To display the form, call the ShowAnnotationForm method, passing in the data object to which to attach the annotation. If the data object that is passed does not support an annotation form, such as if it is not a top-level object, a trappable error is raised. You can also pass the form over which the annotation form appears. This parameter is optional and if nothing is passed, the form appears in the center of the screen. The annotation form appears modally and allows the user all the options available in the program. After you finish using any annotation forms in your project, call the CloseDown method to release all the resources being used by this process.

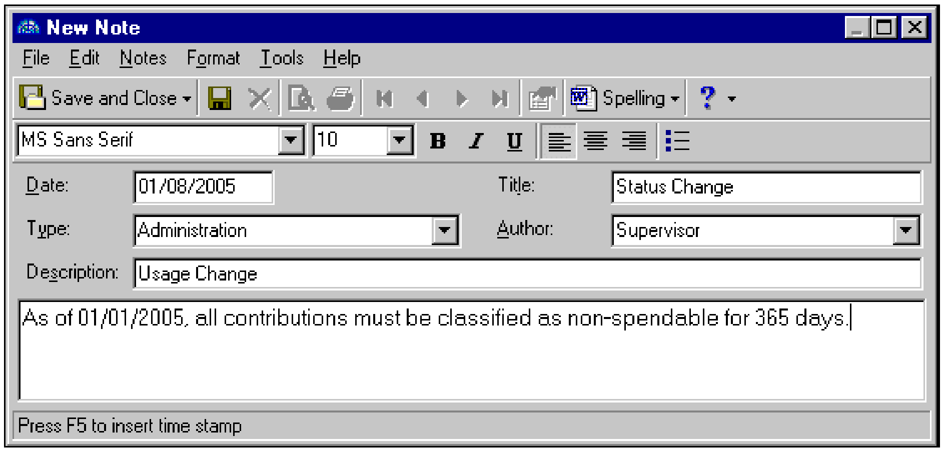
You cannot save an edited annotation to the database until you call the Save method for the data object.

The following code sample illustrates creating an annotation form:

```
Dim FEService As FE_Services
Set FEService = New FE_Services
FEService.Init FE_Application.SessionContext

Dim oAnnotationForm As CAnnotationForm
Set oAnnotationForm = FEService.CreateServiceObject(bbsoAnnotationForm)
oAnnotationForm.Init FE_Application.SessionContext

Dim oProject As CGLProject
Set oProject = New CGLProject

oProject.Init FE_Application.SessionContext
oProject.LoadByField uf_Project_ProjectID, 6
oAnnotationForm.ShowAnnotationForm oProject, Me

'Any changes that the user made on the Annotation Form
'are not saved until this is called.
oProject.Save

'Clean up.
oAnnotationForm.CloseDown
Set oAnnotationForm = Nothing

oProject.CloseDown
Set oProject = Nothing
```

Using Notepad Forms

In ***The Financial Edge***, you can use some top-level data objects to enter multiple notes for each record. These notes are all added via a common form — the Notepad Form service object.



Date: 01/08/2005
Title: Status Change
Type: Administration
Author: Supervisor
Description: Usage Change

As of 01/01/2005, all contributions must be classified as non-spendable for 365 days.

Press F5 to insert time stamp

➢ Using the Notepad Form object

1. Declare an object reference for the FE_Services object. Then create a new instance of the FE_Services object and use the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext to initialize it.
2. Declare an object reference for the CNotepadForm object, then create a new instance of the CNotepadForm object.
3. Set the CNotepadForm object equal to the FE_Services.CreateServiceObject method, passing the enum constant bbsoNotepadForm.
4. Initialize the CNotepadForm object using the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext.
5. Before you can use the Notepad Form object, set the NotepadObjects property to a valid collection. To display an existing note, you can set the NotepadObjectID equal to the database ID of the note you want to display. If this property is not set, a blank form appears and new note is created when the you save the record.
6. Notepad Forms differ from other user interface forms in that you can select a form caption. Set the FormCaption property equal to the string you want to appear, then when the form appears, the caption reads “Notepad For” concatenated with the string you provided.
7. To display the Notepad Form, call the ShowForm method. When a user enters data on the form and selects to save the notepad, the notepad information is not automatically saved to the database. To save it, call the parent record’s Save method.

The following table lists parameters for the ShowForm method of CNotepadForm:

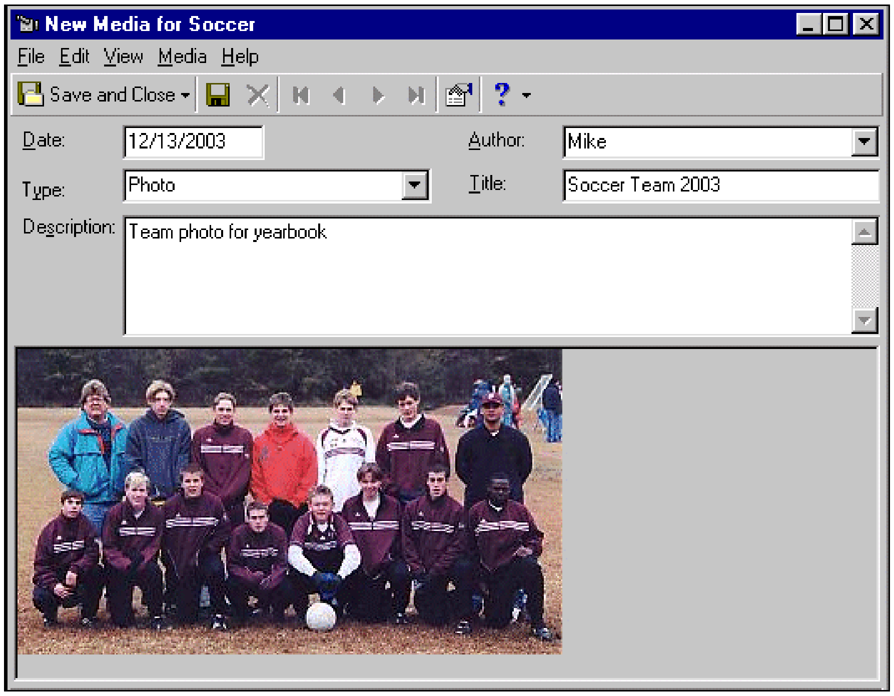
Parameter	**Variable Type**	**Description**
oFormToCenterOn	Object	This is the object over which the Notepad Form appears.
oMoveServer	IBBMoveServer	Optional: This establishes how the “VCR” buttons on the form function. Options are explained in detail in the Programming Reference.
bOKCancel	Boolean	Optional: If set to True (False is the default), the only options under the **File** menu are **Save**, **Save and Close**, **Properties** and **Close**.
bViewOnly	Boolean	Optional: If set to True (False is the default), the user can view notepad information but not edit it.

The following code sample illustrates creating a new notepad form.

```
Dim FEService As FE_Services
Set FEService = New FE_Services
FEService.Init FE_Application.SessionContext

Dim oProject As CProject
Set oProject = New CGLProject
oProject.Init FE_Application.SessionContext
oProject.LoadByField uf_Project_ProjectID, 6

Dim oNotepadForm As CNotepadForm
Set oNotepadForm = FEService.CreateServiceObject(bbsoNotepadForm)
oNotepadForm.Init FE_Application.SessionContext

Set oNotepadForm.NotepadObjects = oProject.Notepads

'If this is not set, a new Notepad is created.
oNotepadForm.NotepadObjectID = oProject.Notepads.Item(1).Fields(NOTEPAD_fld_Id)

'The caption on the form reads "Notepad for " & ProjectDescription
oNotepadForm.FormCaption = oProject.Fields(GLPROJECTS_fld_DESCRIPTION)

'In this case, the form will displayed modally, with all File menu
'options and allow the user to edit the Note.
oNotepadForm.ShowForm Me, True, True, , False, False

'The user's changes are not added to the database until this called.
oProject.Save

'Clean Up!
oProject.CloseDown
Set oProject = Nothing

oNotepadForm.CloseDown
Set oNotepadForm = Nothing
```

Using Media Forms

In ***The Financial Edge***, you can use the Media tab of various records to store media files such as documents, bitmaps (graphics), and video. With the Media Form object, you can incorporate that functionality into your custom applications.



Date: 12/13/2003
Author: Mike
Type: Photo
Title: Soccer Team 2003
Description: Team photo for yearbook

➢ Using the Media Form object

1. Declare an object reference for the FE_Services object. Then create a new instance of the FE_Services object and use the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext to initialize it.
2. Declare an object reference for the CMediaForm object, then create a new instance of the CMediaForm object.
3. Set the CMediaForm object equal to the FE_Services.CreateServiceObject method, passing the enum constant bbsoMediaForm.
4. Initialize the CMediaForm object using the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext.
5. Set the CMediaForm.MediaObjects property equal to a valid collection of media objects, such as CGLProject.Media.
6. At this point, the Media Form object is ready to be used. To display an existing media item, set the MediaObjectID equal to the database ID of the media item you want to display. If this property is not set to anything, a blank form appears and a new media item is created when the record is saved.

7. One area where the Media Form differs from some of the other user interface forms is that you can select the caption of the form when it is appears in the program. Set the NameForCaption property equal to the string you want to appear. When the form appears, the caption is “Media For” concatenated with the string you have passed.

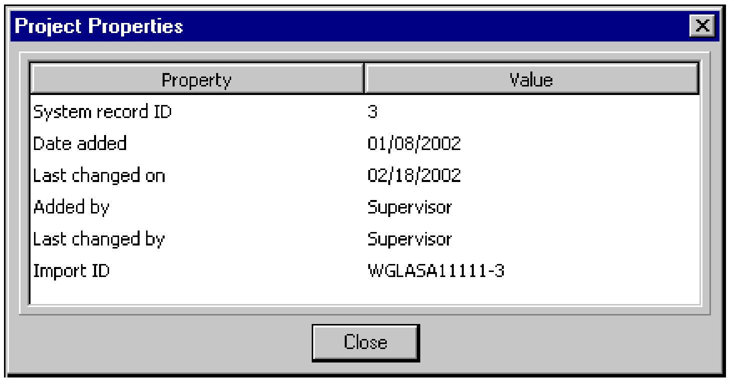
Parameter	Variable Type	Description
oFormToCenterOn	Object	This is the object over which the Media Form appears.
oMoveServer	IBBMoveServer	Optional: This establishes how the “VCR” buttons on the form function. Options are explained in detail in the Programming Reference.

8. To display the Media Form, call the ShowForm method. When a user enters data into the form and selects to save the media item, information is not automatically saved to the database. To save it, call the parent record’s Save method.

```
Dim FEService As FE_Services
Set FEService = New FE_Services
FEService.Init FE_Application.SessionContext

Dim oProject As CGLProject
Set oProject = New CGLProject
oProject.Init FE_Application.SessionContext
oProject.LoadByField uf_Project_ProjectID, 6

Dim oMediaForm As CMediaForm
Set oMediaForm = FEService.CreateServiceObject(bbsoMediaForm)
oMediaForm.Init FE_Application.SessionContext

Set oMediaForm.MediaObjects = oProject.Media

'If this is not set, then a new Media item will be created.
oMediaForm.MediaObjectID = oProject.Media.Item(1).Fields(MEDIA_fld_ID)

'The caption on the form will read "Media for " & ProjectDescription
oMediaForm.NameForCaption = oProject.Fields(GLPROJECTS_fld_DESCRIPTION)

'In this case, the form is displayed modally.
oMediaForm.ShowForm Me, True, True,

'The user's changes are not added to the database until this is called.
oProject.Save

'Clean Up!
oProject.CloseDown
Set oProject = Nothing

oMediaForm.CloseDown
Set oMediaForm = Nothing
```

Using Property Viewers

When using ***The Financial Edge***, various statistics are maintained "behind the scenes". For example, when you create an account record, the date and the creator's user name are stored in the database. To see this information, from the menu bar, select **File**, **Properties**. The Account Properties screen appears, displaying properties of the account record type. The fields shown on the property form vary depending on the record type. Using the Property Viewer service object, you can display this form in your applications.


Project Properties

Property	Value
System record ID	3
Date added	01/08/2002
Last changed on	02/18/2002
Added by	Supervisor
Last changed by	Supervisor
Import ID	WGLASA11111-3

➢ Using the Property Viewer object

1. Declare an object reference for the FE_Services object. Then create a new instance of the FE_Services object and use the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext to initialize it.
2. Declare an object reference for the IBBPropertyViewer object, then create a new instance of the IBBPropertyViewer object.
3. Set the IBBPropertyViewer object equal to the FE_Services.CreateServiceObject method, passing the enum constant bbsoPropertyViewer.
4. Initialize the IBBPropertyViewer object using the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext.
5. Display the properties form by calling the ShowForm method, passing the data object. You can also select the form over which to center the Properties form.

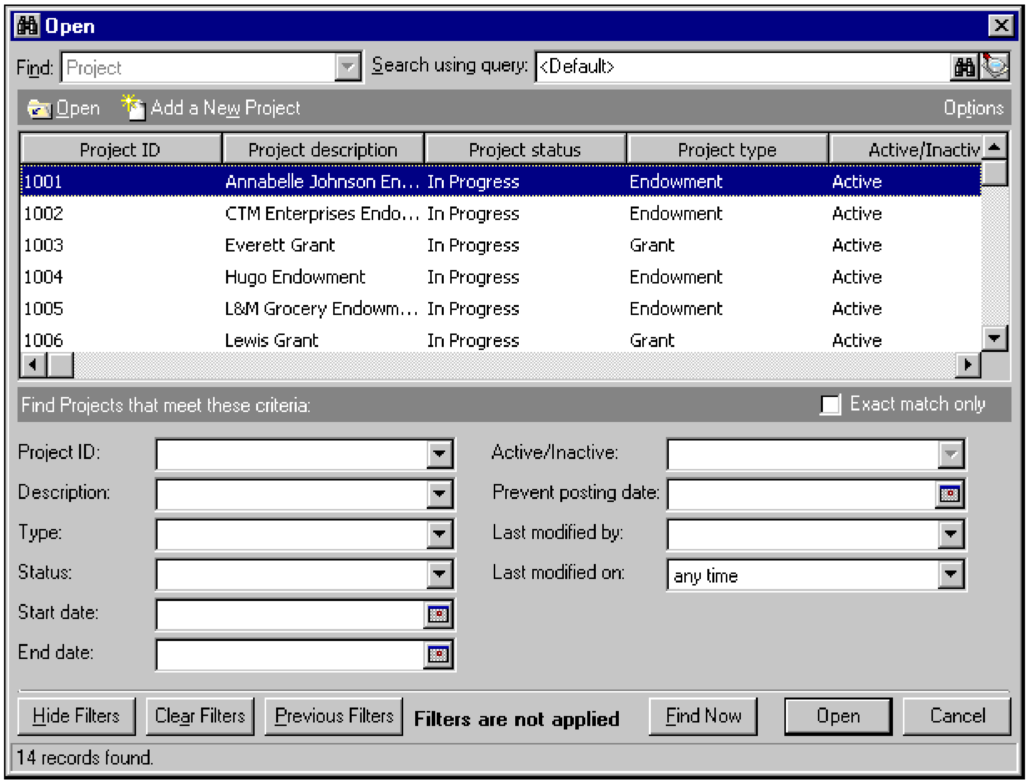
The following code sample displays creating a properties viewer form:

```
Dim FEService As FE_Services
Set FEService = New FE_Services
FEService.Init FE_Application.SessionContext

Dim oAccount As CGlAccount
Set oAccount = New CGlAccount

oAccount.Init FE_Application.SessionContext
oAccount.LoadByField uf_Account_AccountNumber, 6

Dim oPropertyViewer As IBBPropertyViewer
Set oPropertyViewer = FEService.CreateServiceObject(bbsoPropertyViewer)
oPropertyViewer.Init FE_Application.SessionContext

'This will display the properties for the account.
'If the object that you pass doesn't
'support Properties, a trappable error is raised.
oPropertyViewer.ShowPropertyForm oAccount, Me

'Clean Up!
oAccount.CloseDown
Set oAccount= Nothing

oPropertyViewer.CloseDown
Set oPropertyViewer = Nothing
```

When you create the Property Viewer object, you can reuse it to display the Properties form for any type of record. You may want to place the code that creates a new instance of the IBBPropertyViewer in your Form_Load, so you can display the Properties form anywhere on your custom form.

Using Search Screens

The search screen appears extensively throughout ***The Financial Edge***. For example, to open a specific record or select a query for a report, users access the search screen. Using the Search Screen object, you can incorporate this functionality into your project. Search criteria and filters change automatically based on the type of record, and you can add functionality for creating a new record.


Open

Find: Project Search using query: <Default>

Open Add a New Project Options

Project ID	Project description	Project status	Project type	Active/Inactiv
1001	Annabelle Johnson En…	In Progress	Endowment	Active
1002	CTM Enterprises Endo…	In Progress	Endowment	Active
1003	Everett Grant	In Progress	Grant	Active
1004	Hugo Endowment	In Progress	Endowment	Active
1005	L&M Grocery Endowm…	In Progress	Endowment	Active
1006	Lewis Grant	In Progress	Grant	Active

Find Projects that meet these criteria: Exact match only

Project ID:		Active/Inactive:	
Description:		Prevent posting date:	
Type:		Last modified by:	
Status:		Last modified on:	any time
Start date:			
End date:			

Hide Filters Clear Filters Previous Filters **Filters are not applied** Find Now Open Cancel

14 records found.

➢ **Using the Search Screen object**

1. Declare an object reference for the FE_Services object. Then create a new instance of the FE_Services object and use the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext to initialize it.
2. Declare an object reference for the IBBSearchScreen object, then create a new instance of the IBBSearchScreen object.
3. Set the IBBSearchScreen object equal to the FE_Services.CreateServiceObject method, passing the Enum constant bbsoSearchScreen.
4. Initialize the IBBSearchScreen object using the Init method (remember to call the CloseDown method when finished), passing a valid SessionContext.

   Because the search screen may be used multiple times in a single project, you may want to declare this as a global- or module-level reference. You can call the Init when the form loads or when it is first used and then call the CloseDown when the form unloads.
5. Before you display the search screen, you need to tell it the types of records you want to be available in the **Find** field. Call the AddSearchType method to add at least one search type before displaying the form. To have more than one type of record available, there are two syntax styles supported. If you have used the object previously, you can call the ClearSearchTypes method to clear any existing search types.

The following code samples show two methods for determining records to include in the search:

```
'Method 1
oSearchScreen.AddSearchType SEARCH_GLACCOUNT
oSearchScreen.AddSearchType SEARCH_GLACCOUNTCODE
oSearchScreen.AddSearchType SEARCH_GLPROJECT

'Method 2
oSearchScreen.AddSearchType SEARCH_GLACCOUNT,SEARCH_GLACCOUNTCODE,SEARCH_GLPROJECT
```

Before you display the search screen form, you can set optional properties for the form. If you set the AllowAddNew property to True, an **Add New** button appears on the form. You must write the code that actually creates the new record. If you add multiple search types, you can set the DefaultSearchType property to determine the search type to default to when the form appears.

To display the search screen, call the ShowSearchForm method. This displays the form modally and returns False if the user clicks **Cancel**. When the user clicks any button that closes the form (**Open**, **Cancel**, or **Add New**), the SelectedOption property returns the button the user selected. The Enum bbSearchScreenOption lists options, simplifying your programming. If multiple search types exist, the SelectedSearchType property determines the search type the user selected. This returns a value from the Enum bbSearchTypes. The SelectedDataObject property returns a reference to the actual data object the user selected.

The following code sample illustrates adding a search screen:

```
Private Sub UsingTheSearchScreen()

    Dim FEService As FE_Services
    Set FEService = New FE_Services
    FEService.Init FE_Application.SessionContext

    Dim oSearchScreen As IBBSearchScreen
    Set oSearchScreen = FEService.CreateServiceObject(bbsoSearchScreen)
    oSearchScreen.Init FE_Application.SessionContext

    With oSearchScreen
        .ClearSearchTypes
        .AddSearchType SEARCH_GLACCOUNT,SEARCH_GLACCOUNTCODE,SEARCH_GLPROJECT
        .DefaultSearchType = SEARCH_GLACCOUNT
        .AllowAddNew = True

        If .ShowSearchForm Then
            Select Case .SelectedSearchType

                Case SEARCH_GLACCOUNT
                    Dim oAccount as CGLAccount

                    Dim oAccountForm As CGLAccountForm
                    Set oAccountForm= New CGLAccountForm
                    oAccountForm.Init FE_Application.SessionContext

                    If oSearchScreen.SelectedOption = SRCH_FRM_OPEN Then
                        Set oAccount = oSearchScreen.SelectedDataObject
                    Else
                        Set oAccount = New CGLAccount
                        oAccount.Init FE_Application.SessionContext
                    End If

                    Set oAccountForm.AccountObject = oAccount
                    oAccountForm.ShowForm True, Me, True

                    oAccountForm.CloseDown
                    Set oAccountForm= Nothing
                    oAccount.CloseDown
                    Set oAccount = Nothing

                Case SEARCH_GLACCOUNTCODE
                    'Same as above except substitute Account Code objects

                Case SEARCH_GLProject
                    'Same as above except substitute Project objects
            End Select

        End If

    End With
```

(Continued- page 2 of 2)

```
    'Clean Up!
    oSearchScreen.CloseDown
    Set oSearchScreen = Nothing

    FEService.CloseDown
    Set FEService = Nothing

End Sub
```

Using the Search Screen object properly in your custom applications not only simplifies coding but also provides your users with a common interface for selecting records. Other methods and properties for this form are explained in the Programming Reference section of the VBA and API help file.

Managing Transactions

In ***The Financial Edge*** object model, a number of collections exist that support the use of transactions. Any object that implements IBBDataObject supports transactions. With transactions, you can add or remove any number of items from a collection, or you can make temporary changes to fields in a data object until you can make those changes permanent. At any point in a transaction, you can select to undo the changes you made. Three methods and one property work together to provide transaction functionality.

The BeginCollectionTransaction (or BeginFieldsTransaction for data objects) method signals the collection or object you want to begin a transaction at this point. If you later select to undo the changes, the collection or object returns to this state.

Calling the CommitCollectionTransaction (or CommitFieldsTransaction for data objects) method tells the collection or object you are finished with this transaction, and you want to save any changes made after the BeginCollectionTransaction. However, this does not make any changes to the database itself — that happens only after calling the parent record’s Save method.

To return the collection or object to the state it was in when you started the transaction, you can call the RollbackCollectionTransaction (or RollbackFieldsTransaction for data objects) method. After calling this method, the collection or object returns to the state it was in when the transaction began.

When using collections, you can use the InTransaction property to see if the collection is in the middle of a transaction. This is important because, if you call CommitCollectionTransaction or RollbackCollectionTransaction when there is no active transaction, an error is raised.

The following sample illustrates removing contacts from a collection within a transaction.

```
Dim oProject As CGLProject
Set oProject = New CGLProject

oProject.Init FE_Application.SessionContext
oProject.LoadByField uf_Project_ProjectID, 3

Dim oContacts As CGLProjectContacts
Set oContacts = oProject.Contacts

oContacts.BeginCollectionTransaction
oContacts.Remove oContacts.Item(1), False

'If we just removed the last Individual relation then Rollback.
If oContacts.Count = 0 Then
    oContacts.RollbackCollectionTransaction
Else
    oContacts.CommitCollectionTransaction
End If

'Any changes to the collection are not saved to the database until now.

oProject.Save
Set oContacts = Nothing

oProject.CloseDown
Set oProject = Nothing
```

Blackbaud VBA

Chapter 3

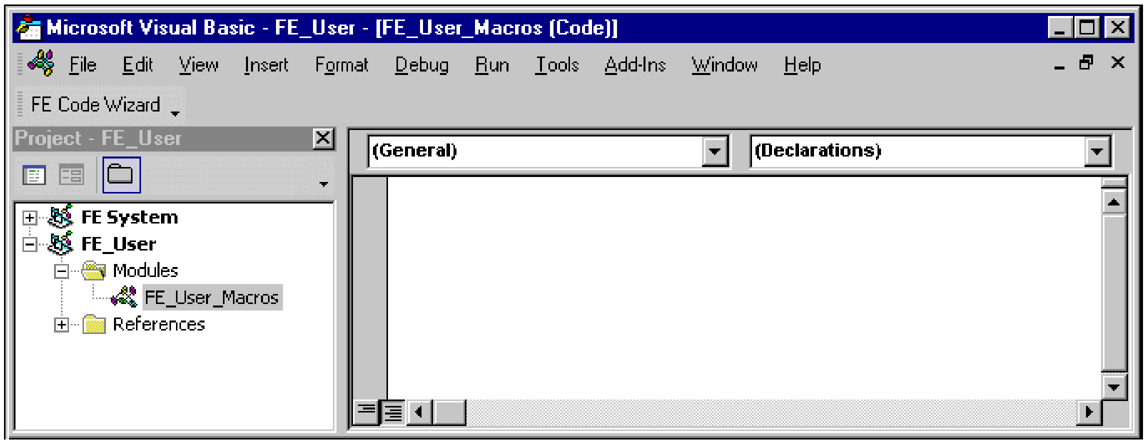
Contents

This chapter introduces using *Visual Basic for Applications* (VBA) in ***The Financial Edge***. With the optional *VBA* module, you can use macros to leverage the power of our software programmatically from within the program shell.

The programming examples and related code provided in this document are the property of Blackbaud, Inc., and you may not copy, distribute, convey, license, sublicense, or transfer any rights therein. All code samples are subject to applicable copyright laws.

Working in the VBA Environment

Before you begin working with *Visual Basic for Applications*, you should understand that VBA is "hosted" by ***The Financial Edge***. VBA is not a standalone application, but rather is an optional module that fully integrates with your Blackbaud software. Many other applications, such as Microsoft *Excel* and *Word*, also host their own instances of VBA. While all these hosts share the common VBA-integrated development environment (IDE), each exposes its own custom functionality and documents as project items. To access the VBA IDE, from the menu bar of the program shell, select **Tools**, **Visual Basic for Applications**. The VBA IDE appears.

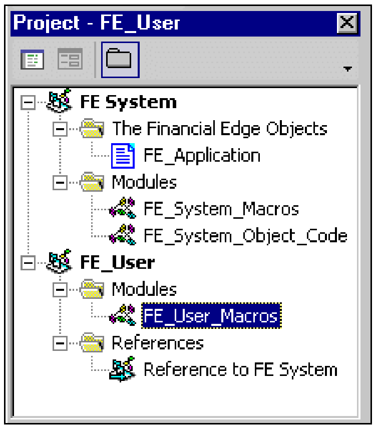

Microsoft Visual Basic - FE_User - [FE_User_Macros (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
FE Code Wizard
Project - FE_User
FE System
FE_User
Modules
FE_User_Macros
References
(General)
(Declarations)

The integrated development environment, which runs in its own window, contains advanced debugging features, property and code editing features (including compile time syntax checking), an enhanced object browser, and code organization and tracking. These features make the VBA IDE a powerful platform from which to develop code. Another useful feature of the *Visual Basic for Applications* environment is that it has the same look and feel regardless of the VBA host application you use, so VBA has the same tools and layout as VBA in other applications such as *Word* or *Excel*.

Like most other development environments, the VBA environment consists of a series of windows that perform specific functions. In VBA, you use the Project Explorer, Code, Properties, and Forms Designer windows to build your VBA applications.

The Project Window

The Project window, also known as the “Project Explorer”, organizes all VBA code stored within the program. ***The Financial Edge*** has two default project items: System and User. The Project window appears automatically when you first access *VBA*. You can also open it from the menu bar by selecting **View**, **Project Explorer**.

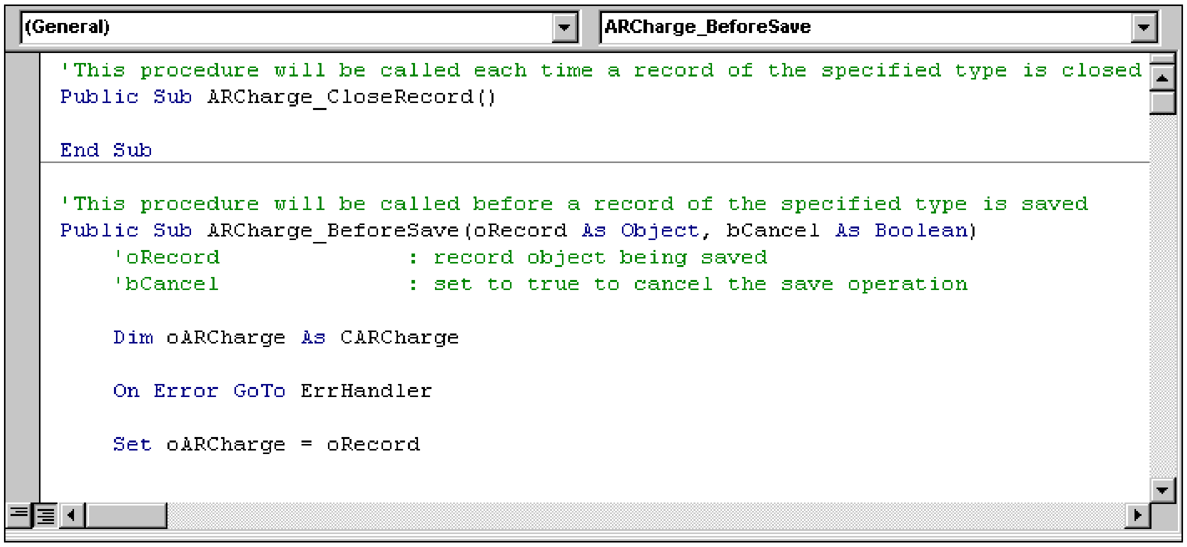


The FE_System project holds references to active instances of objects in ***The Financial Edge***, and it contains code that executes for all users. This project is the place to build your custom business rules and functions. Only the supervisor can edit code in the system project. The code executes for other users, but they cannot modify it.

The FE_User project gives individual end users a project in which to save individual code and macros. Code stored in the User project is available to a specific user. By isolating user projects, you can keep code organized and prevent tampering with your important system code.

The Code Window

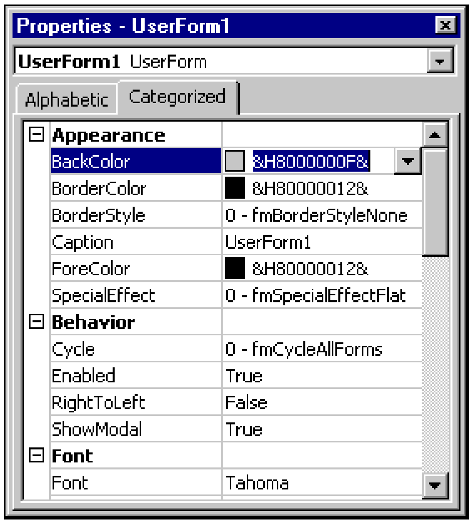
The code window is a text editor in which you view, edit, and debug code. To view a project's code, in the Project Explorer, select an element containing code. Then on the Project Explorer toolbar, click **View Code**. The code appears.



```
'This procedure will be called each time a record of the specified type is closed
Public Sub ARCharge_CloseRecord()

End Sub

'This procedure will be called before a record of the specified type is saved
Public Sub ARCharge_BeforeSave(oRecord As Object, bCancel As Boolean)
    'oRecord              : record object being saved
    'bCancel              : set to true to cancel the save operation

    Dim oARCharge As CARCharge

    On Error GoTo ErrHandler

    Set oARCharge = oRecord
```

The Properties Window

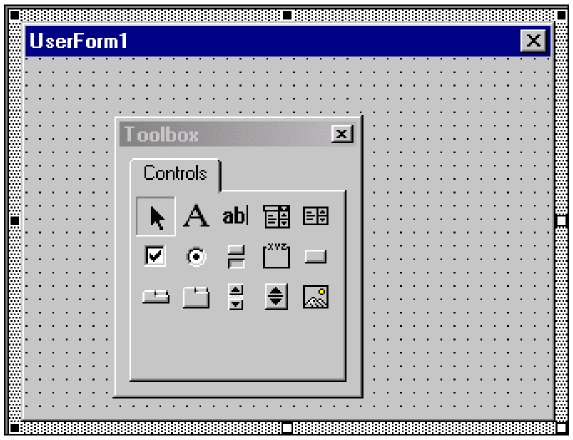
A property is an attribute of an object, such as its color or caption. Using the Properties window, you set properties to specify characteristics or the behavior of an object at design time. This window is useful when building custom dialog boxes and forms using VBA's forms designer. To access the Properties window, on the menu bar select **View**, **Properties Window**.



⊟ **Appearance**	
BackColor	&H8000000F&
BorderColor	&H80000012&
BorderStyle	0 - fmBorderStyleNone
Caption	UserForm1
ForeColor	&H80000012&
SpecialEffect	0 - fmSpecialEffectFlat
⊟ **Behavior**	
Cycle	0 - fmCycleAllForms
Enabled	True
RightToLeft	False
ShowModal	True
⊟ **Font**	
Font	Tahoma

The Forms Designer Window

VBA provides full-featured forms design support through the Forms Designer. After you create forms, you can call them from VBA code you write to execute within ***The Financial Edge***. For example, you can design a custom data entry form that appears when an end user saves a project record. To access the Forms Designer, from the menu bar select **Insert**, **UserForm**.



Managing Active Objects

With Active Objects, you can respond to events within ***The Financial Edge***. For example, you can build custom business rules that are applied before the program can save or delete a record. There are three types of active objects: the FE_Application object, which you can use to alter the program's opening and closing procedures; active data objects, which you can use to manipulate top-level objects such as invoices and accounts; and active process objects, which control processes such as reports and imports.

Active objects are divided into three groups:

- The FE_Application object represents the application. Here you can write code that runs when a user opens or closes the application. To cancel the close event, use the bCancel parameter.
- Data objects include all top-level objects. Examples include GLAccount, APVendor, ARPayment, FAAsset, PYEmployee, and Banks. These objects all have events tied to opening, closing, saving, and deleting a record.
- Process objects include ActiveImport, ActiveMail, and ActiveReport. These objects have events for starting and ending a process. In addition, with *Import* you can interact with each record while it is validated.

Managing the FE_Application Object

The FE_Application object represents the overall program. It provides access to a valid SessionContext, which you use to initialize ***Financial Edge*** objects. The FE_Application object also provides two events for which you can write code. The UIOpening event fires as the shell opens and the UIClosing event fires as the shell is closed. The FE_Application object also has one read-only property, Version, which returns the complete version and build number of ***The Financial Edge***. For information about accessing the SessionContext from VBA, see "Initializing and Releasing Objects" on page 14.

UIOpening Event

Using the UIOpening event, you can execute code when a user begins a ***Financial Edge*** session. For example, you can create a reminder that appears when a user logs in or connects to another application's database to transfer data between applications. The UIOpening event fires immediately after the shell appears on the screen, but before the Home page appears.

UIClosing Event

Using the UIClosing event, you can execute a section of code just before the program completely closes. It gives you the chance to close down a connection to another application's database or to ensure the user has correctly completed a specific task. With the UIClosing event, you can also prevent the shell from closing by setting the bCancel variable to True. This stops the program from closing and returns the user to the program shell.

Managing Active Data Objects

Active data objects, the most common objects, are provided for each top-level object. An active object's name begins with the module that contains that object. For example, the GLAccount object pertains to *General Ledger*. The APInvoice object pertains to *Accounts Payable*. When creating macros, you can run the Code Wizard to select an event. The Code Wizard then displays the object's code so you can place your custom code in the object's events. When one of these events fires, the code you wrote for the event executes automatically. To view or edit active data objects, you must have Supervisor rights in the program.

Using an active object's events, you can:

- Notify users of required fields and prevent them from saving records without data in these fields.
- Prevent users from deleting records meeting certain criteria. For example, you could prevent deleting projects with an active status.
- Maintain a separate database with information entered into or generated by ***The Financial Edge***.
- Send email to specific individuals at your organization when certain conditions are met. For example, you can notify all members of the accounting department when an account is marked inactive.

The BeforeOpen Event

The BeforeOpen event fires before the user interface form for a new or existing record appears. This gives you the opportunity to perform any pre-processing to the underlying data object or to display instructions for your users.

The BeforeOpen event passes the following parameter:

- oRecord, an object representing the active object. The oRecord object is already initialized, so there is no need to call the Init method or the CloseDown method at the end of your subroutine. Using the oRecord object, you can set properties such as fields or you can use the object's methods before the form appears to your user.

Insert the following code in the BeforeOpen event of the GLProject object. It notifies users they are editing an inactive project.

```
Private Sub GLProject_BeforeOpen(oRecord As Object)

    Dim sMessage As String
    Dim lResult As Long
    Dim oProject As CGLProject

    'Setting oProject = oRecord is not necessary
    'but it activates IntelliSense for the object
    Set oProject = oRecord

    If oProject.Fields(GLPROJECTS_fld_ACTIVEFLAG) = "Inactive - prevent data entry" Then

        sMessage = "Project " & oProject.Fields(GLPROJECTS_fld_PROJECTID)
        sMessage = sMessage & " is currently marked inactive. Would you like to"
        sMessage = sMessage & " reactivate the project?"

        lResult = MsgBox(sMessage, vbYesNo, "Inactive Project")

        If lResult = vbYes Then
            oProject.Fields(GLPROJECTS_fld_ACTIVEFLAG) = "Active"
        End If

    End If

    Set oProject = Nothing

End Sub
```

The BeforeSave Event

The BeforeSave event fires before the program performs any Save operation (Save, Save and New, or Save and Close). With this event, you can enforce any business rules specific to your organization.

The BeforeSave event passes the following parameters:

- oRecord, which represents the active object. The oRecord object is already initialized, so there is no need to call the Init method or the CloseDown method at the end of the subroutine.
- bCancel that, if set to True, cancels the save operation and returns focus to the active form.

This example ensures that a project ID is a four-digit numeric value. You would place this code in the BeforeSave() event of the Active_GLProject object.

After the code runs, ***The Financial Edge*** still performs its usual validation to guarantee all required fields are completed and date fields contain valid dates.

```
Private Sub GLProject_BeforeSave(oRecord As Object, bCancel As Boolean)

    Dim oProject As CGLProject
    Dim 1ID 'ProjectID

    'Setting oProject = oRecord is not necessary
    'but it activates IntelliSense for the object
    Set oProject = oRecord

    lID = oProject.Fields(GLPROJECTS_fld_PROJECTID)

    If Len(lID) = 4 Then
        If IsNumeric(lID) Then
            Set oProject = Nothing
            Exit Sub
        Else
            MsgBox "Project ID must be numeric"
            bcancel = True
        End If
    Else
        MsgBox "Project ID must be a 4 digit number"
        bcancel = True
    End If

    Set oProject = Nothing

End Sub
```

The AfterSave Event

The AfterSave event fires after data has been written to the database. In the following example, we use the AfterSave event to determine if the saved project is a Scholarship type. If it is, Scholarship.mdb is updated, assuming the record does not already exist. Also, the program sends an email to the controller to provide information about the new scholarship entry. You would place this code in the AfterSave() event of the GLProject object.

To access objects used in this email code sample, you must add a reference to the Microsoft Outlook Object Library.

The event has one parameter, oRecord, representing the active data object. In this case, the oRecord is of type CGLProject.

```
Private Sub GLProject_AfterSave(oRecord As Object)

    Dim oProject As CGLProject
    Dim lProjectID As Long

    Dim oMDB As Database
    Dim oScholarshipFund As Recordset

    'Setting the oProject = oRecord is not necessary
    'but it activates IntelliSense for the object
    Set oProject = oRecord

    'If this is a 'scholarship' type project, proceed
    If oProject.Fields(GLPROJECTS_fld_TYPE) = "Scholarship" Then

        lProjectID = oProject.Fields(GLPROJECTS_fld_PROJECTID)

        Set oMDB = OpenDatabase("c:\Scholarship.mdb")
        Set oScholarshipFund = oMDB.Recordset("ScholarshipFund")

        With oScholarshipFund

            .Index = "PrimaryKey"
            .Seek "=", lProjectID

            '.NoMatch will = True if the record is not found
            'If it is found then set objects = nothing and exit the subroutine
            If Not .NoMatch Then
                Set oProject = Nothing

                oScholarshipFund.Close
                Set oScholarshipFund = Nothing

                oMDB.Close
                Set oMDB = Nothing

                Exit Sub
            End If

            'If a match is not found, add the record
            .AddNew
            .Fields("DatabaseID") = oProject.Fields(GLPROJECTS_fld_GL7PROJECTSID)
            .Fields("ProjectID") = oProject.Fields(GLPROJECTS_fld_PROJECTID)
            .Fields("ProjectDescription") = oProject.Fields(GLPROJECTS_fld_DESCRIPTION)
            .Update

            .Close
            Set oMDB = Nothing

        End With
```

(Continued- page 2 of 2)

```
        'send the email message
        Dim oOutlook As Outlook.Application
        Set oOutlook = New Outlook.Application

        Dim oMailItem As MailItem
        Set oMailItem = oOutlook.CreateItem(olMailItem)

        With oMailItem
            .To = "Controller@YourOrganization.com"
            .Subject = "New Scholarship Project"
            .Body = "Project " & oProject.Fields(GLPROJECTS_fld_PROJECTID) & " - " & _
                    oProject.Fields(GLPROJECTS_fld_DESCRIPTION) & " has been added to " & _
                    "the Scholarship database."
            .Display
            'Uncomment the following line if you want the email to be
            'sent without user intervention. Comment '.display' above
            '.Send
        End With

        Set oMailItem = Nothing
        Set oOutlook = Nothing

        MsgBox "Scholarship database has been updated.", vbOKOnly, "FE 7"

    End If
End Sub
```

The CloseRecord Event

The CloseRecord event fires just before the active data object and user interface form closes. This gives you the opportunity to close down any objects you may have opened when using this record.

If you need to update an external database, you may want to add connection information to the BeforeOpen event and disconnection information to the CloseRecord event.

The BeforeDelete Event

The BeforeDelete event fires just before the active data object is deleted. Using the BeforeDelete event, you can check any business rules specific to your organization and cancel the delete process if necessary.

The BeforeDelete event passes the following parameters:

- oRecord, which represents the active object. It is already initialized, so you do not need to call the Init method or the CloseDown method at the end of the subroutines.
- bCancel that, if set to True, cancels the delete operation and returns focus to the active form.

In this example, we check the project record to ensure the End Date is not in the future. You would place this code in the BeforeDelete event of the GL_ActiveProject record.

```
Private Sub FERecord_BeforeDelete(oRecord As Object, bcancel As Boolean)

    Dim oProject As CGLProject
    'bInvalid Date returns 'true' if Project End Date is in the future
    Dim bInvalidDate As Boolean

    'Setting the oProject = oRecord is not necessary
    'but it activates IntelliSense for the object
    Set oProject = oRecord

    Select Case oproject.Fields(GLPROJECTS_fld_ENDDATE)
        Case Is > Date
            bInvalidDate = True
        Case Else
            bInvalidDate = False
    End Select

    If bInvalidDate Then
        bcanel = (MsgBox("The Project End Date is in the future. " & _
                "Would you like to delete the project anyway?", vbYesNo, "FE 7")=vbNo)
    End If

    Set oProject = Nothing

End Sub
```

The AfterDelete Event

The AfterDelete event fires after a record is deleted. The event passes both the Record ID and the Import ID from the deleted record. Both of these fields are unique and provide a way to identify the deleted record.

This example opens the Microsoft *Access* database named Scholarship.mdb and deletes a record if it exists. You would place this code in the AfterDelete event of the GL_ActiveProject object.

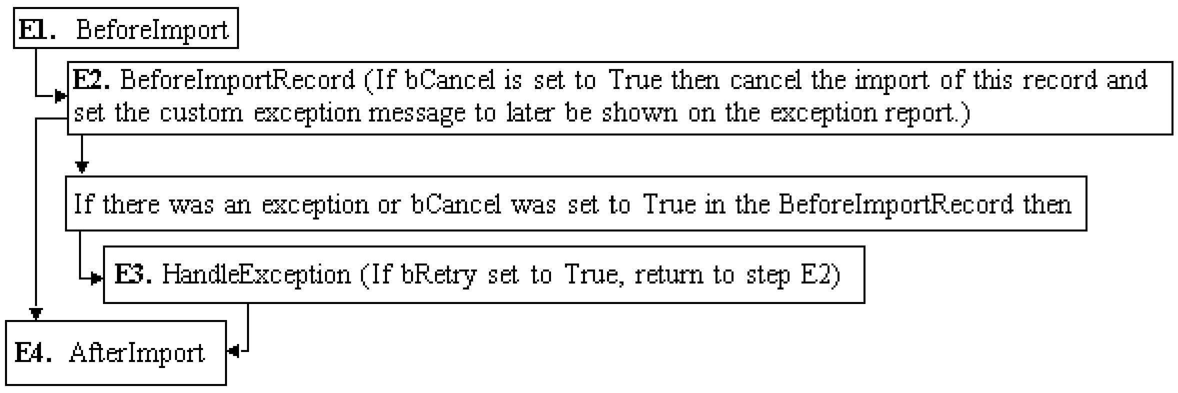
```
Private Sub FERecord_AfterDelete(lDatabaseID As Long, sImportID As String)

    Dim oMDB As Database
    Dim oScholarshipFund As Recordset

    Set oMDB = OpenDatabase("c:\Scholarship.mdb")
    Set oScholarshipFund = oMDB.Recordset("ScholarshipFund")

    With oScholarshipFund
        .Index = "PrimaryKey"
        .Seek "=", lDatabaseID

        If Not .NoMatch Then .Delete
    End With

    oScholarshipFund.Close
    Set oScholarshipFund = Nothing

    oMDB.Close
    Set oMDB = Nothing

End Sub
```

Managing Active Process Objects

In addition to top-level data objects, in VBA you can access and control certain processes. Using service objects, you have access to import, mail, post, and report processes. The specific events provided vary based on the process, but in general they enable you to perform actions before and after the process and to handle exception processing. Import objects such as BeforeImport are specific to *Import*, while process objects such as BeforeProcess are used in both *Reports* and *Mail*.

VBA in *Import* provides four events so you can write code that executes during the importing process. With these events, you can create custom exceptions that prevent unwanted records from entering the database, and you can also specify custom exception messages displayed on reports. You can even change a record that was previously an exception and instruct *Import* to retry the import.

The diagram shows the events and the order in which they fire.


E1. BeforeImport
E2. BeforeImportRecord (If bCancel is set to True then cancel the import of this record and set the custom exception message to later be shown on the exception report.)
If there was an exception or bCancel was set to True in the BeforeImportRecord then
E3. HandleException (If bRetry set to True, return to step E2)
E4. AfterImport

The BeforeImport Event

The BeforeImport event fires once, just before the importing of the records to the database begins. You can start any processes you would like to have in place while importing. For example, if you are also importing some of this data into a separate database, you can use this event to connect to that database.

The BeforeImport event passes the following parameters:

- sImportName, a string representing the name of the import being processed.
- lImportType from the Enum bbImportTypes, which tell you the import type, such as Account or Project.
- bCancel which, if set to True, discontinues the importing process.

The following code sample opens two text files: one for storing the ID and description of the imported records, the other for records that are exceptions.

```
Public Sub ImportVBARecord_BeforeImport(ByVal sImportName As String, _
                                        ByVal lImportType As FEInterfaces.bbImportTypes, _
                                        ByRef bcancel As Variant)

    'For project import files we need to provide a list of projects
    'imported and those that were exceptions. This routine opens the two text files
    'that will hold the ID and description of the records processed.

    If lImportType = bbImportType_GL_PROJECTS Then

        If MsgBox("Create import summary files?") = vbYes Then
                ' mlExcFile and mlImpFile are module level variables
                ' that hold the two file handles

            mlExcFile = FreeFile
            Open "C:\Exc_" & sImportName & ".IMP" For Append As mlExcFile

            mlImpFile = FreeFile
            Open "C:\Imp_" & sImportName & ".IMP" For Append As mlImpFile
        End If

    End If

End Sub
```

The BeforeImportRecord Event

The BeforeImportRecord event fires just before each row in an import file is imported into the database. You can access the underlying data object after it is created but before it is added to the database. You may use this to send some information to a separate database, or you can use it to verify whether specific business rules for your organization have been met. If they have not been met, you can prevent the record from being imported into the database.

The BeforeImportRecord event passes the following parameters:

- sImportName, a string representing the name of the import being processed.
- ImportType, from the Enum bbVBAImportTypes. This tells you the type of the import, such as Account or Project.
- oDataObject; the type of this variant depends on the import type. You can fully access the entire object model for the oDataObject type at this time. You can change information, use VBA to calculate and add numbers to the record, or verify that all information meets your organization’s business rules.

- bCancel, which if for some reason you do not want a particular record added to the database, you can set equal to True. The record will not be added to the database and is listed as an exception on the Exception Report.
- sExceptionMessage, which you can set equal to the reason for the exception so it prints on the Exception Report and tells your end-user what was wrong with this record. Any time a record is flagged as an exception, the HandleException event fires.

The following code sample illustrates writing the IDs and descriptions of imported records to a text file.

```
Public Sub ImportVBARecord_BeforeImportRecord(ByVal sImportName As String, _
                                         ByVal lImportType As FEInterfaces.bbImportTypes, _
                                         oDataObject As Variant, _
                                         ByRef bCancel As Variant, _
                                         ByRef sExceptionMessage As Variant)

    Dim oProject as CGLProject

    'mlImpFile is a module level variable; a file handle
    'for the imported records file
    If mlImpFile > 0 Then
        Set oProject = oDataObject
        Print #mlImpFile, oProject.Fields(GLPROJECTS_fld_PROJECTID) & ", " & _
                          oProject.Fields(GLPROJECTS_fld_DESCRIPTION)
        Set oProject = Nothing
    End If

End Sub
```

The HandleException Event (Active Import)

The HandleException event fires whenever a record in an import is flagged as an exception. You can change the record to fix the cause of the exception and try again or, if you transferred information for each record to an external database in the BeforeImportRecord event, you may want to remove that information using this event.

The HandleException event passes the following parameters:

- sImportName is a string representing the name of the import being processed.
- lExceptionCode tells you the reason for the exception, from the Enum bbImportExceptionCodes.
- oDataObject; the data type of this variant depends on the import type. You can access the entire object model for the oDataObject at this time.
- bTryAgain, if set to True, causes the program to retry importing the updated record to the database. It is very important that you make sure the cause of the exception has been fixed, so there is no possibility of the user getting caught in a loop.

The following code sample illustrates writing the IDs and descriptions of exception records to a text file.

```
Public Sub ImportVBARecord_HandleException(ByVal sImportName As String, _
                                           ByVal lExceptionCode As _
                                             FEInterfaces.bbImportExceptionCodes, _
                                           oDataObject As Variant,
                                           ByRef bTryAgain As Variant)

    Dim oProject As CGLProject

    'mlExcFile is a module level variable; a file handle for the exception file
    If mlExcFile > 0 Then
        Set oProject = oDataObject
        Print #mlExcFile, oProject.Fields(GLPROJECTS_fld_PROJECTID) & ", " & _
                          oProject.Fields(GLPROJECTS_fld_DESCRIPTION)
        Set oProject = Nothing
    End If

End Sub
```

The AfterImport Event

The AfterImport event fires once after the entire import process is complete. You can clean up any objects or connections to other databases you were using while the import was processing.

The AfterImport event passes the following parameters:

- sImportName, a string representing the name of the import being processed.
- lNumRecsImported, a long that represents the number of records added successfully.
- lNumExceptions, a long that represents the number of records that caused exceptions.

The following code sample illustrates closing the text files and informing the user that the import files now exist.

```
Public Sub ImportVBARecord_AfterImport(ByVal sImportName As String, _
                                       ByVal lNumRecsImported As Long, _
                                       ByVal lNumExceptions As Long)
    'mlExcFile is a module level variable; a file handle for the exception file
    If mlExcFile > 0 Then
        Close mlExcFile
        'mlImpFile is a module level variable; a file handle for the imported records file
        Close mlImpFile

        MsgBox "Import summary files created"
    End If

End Sub
```

The BeforeProcess Event

Reports and *Mail* both support VBA "process" events. For example, the BeforeProcess event fires just before a mail or report function begins to process. You can prevent the report or mail function from processing, and if you are exporting information, you can specify that the name of the export file appear so you can start a word processing merge or automatically graph the information in a spreadsheet.

The BeforeProcess event passes the following parameters:

- lReportType/lMailType is a long that can be used with the Enum Ebbrep_ReportTypes to determine the type of report or mail function being run, such as a General Ledger Report or Balance Sheet.
- sParamName is a string that is the name of the actual parameter file used for this report or mail function.
- lAction is a long that, when used with the Enum Ebbrep_ProcessOptions, tells you whether the user is printing, print previewing, or exporting the report or mail function.
- bCancel, if set to True, cancels the entire process before it starts.

The following code sample limits users to running *General Ledger* reports only after 5:00 p.m., when all data entry is complete for the day.

```
Private Sub ReportsVBARecord_BeforeProcess(ByVal lReportType As Long, _
                                           ByVal sParamName As String, _
                                           ByVal lAction As Long, _
                                           bCancel As Boolean,
                                           ByVal Reserved As Variant)
    Select Case lReportType

        Case bbrep_GL_GeneralLedgerReport

            If Time < #5:00:00 PM# Then
                MsgBox "This report can only be run after 5:00 pm."
                bCancel = True
            End If

    End Select

End Sub
```

The AfterProcess Event

The AfterProcess event fires after the mail or report function finishes processing and displays its output, if any.

The AfterProcess event passes the following parameters:

- lReportType/lMailType, a long that can be used with the Enum Ebbrep_ReportTypes to determine the type of report or mail function that has just run, such as the Balance Sheet or Income Statement.
- sParamName, a string that stores the name of the actual parameter file used for this report or mail function.
- lAction, a long that, when used with the Enum Ebbrep_ProcessOptions, tells you whether the user is printing, print previewing, or exporting the report or mail function.
- sExportFileName, which, if the Action performed was exporting, contains the full path for the export file name.

The following code sample sends an email copy of Today’s Reports to the controller after importing, using the Microsoft Outlook Object Library.

```
Private Sub ReportsVBARecord_AfterProcess(ByVal lReportType As Long, _
                                          ByVal sParamName As String, _
                                          ByVal lAction As Long, _
                                          ByVal sExportInfo As String, _
                                          ByVal Reserved As Variant)

    On Error GoTo ehSendReport

    'You will need to set a reference to Microsoft Outlook 9.x Object Library
    Dim oOutlook As Outlook.Application 'This starts Outlook.
    Dim oMailItem As MailItem

    'If the user is printing the 'Today's Reports' then
    If sParamName = "Today's Reports" Then If

        Len(sExportInfo) > 0 Then
            'Create the objects needed for email
            Set oOutlook = New Outlook.Application
            Set oMailItem = oOutlook.CreateItem(olMailItem)

            With oMailItem
                .To = "Controller@YourOrganization.com"
                .Subject = sParamName
                .Attachments.Add (sExportInfo)
                .Send
            End With

            'Close all object references
            Set oMailItem = Nothing
            Set oOutlook = Nothing
        End If

    End If

    Exit Sub

ehSendReport:

    MsgBox Err.Description, vbOKOnly

End Sub
```

Managing VBA Macros

You can automate task that you perform regularly with macros. A macro is a series of steps stored in a VBA module or VBA *.dll file you can run when you need to perform the task. You can create and edit macros directly through the program’s shell, or you can use a separate program, FE7VBA.exe. The process of creating macros through either means is similar, but each method has its own advantages. For our purposes, macros created through the program’s shell are referred to as VBA Macros, while those created through the FE7VBA.exe program are referred to as VBA DLL Macros.

With VBA, you can create four different types of macros.

Active Object Macros. With Active Object macros, you can control top-level objects or processes at key points in their lifetime. Standard events occur for each data object as it is loaded, saved, closed, and deleted. Other events fire at the beginning and end of certain processes. You can write code inside these events to act on a given data object, to enforce custom business rules, or to prevent processes from occurring under specified conditions.

Standard Macros. With Standard macros, you can perform specific functions from the program shell. For example, you can write a macro that automates the printing of a set of End of Day, End of Month, or End of Year reports. You first create the reports for each group and then create the macro to automate the printing process. Your users can select the macro and run all the reports with the click of a button.

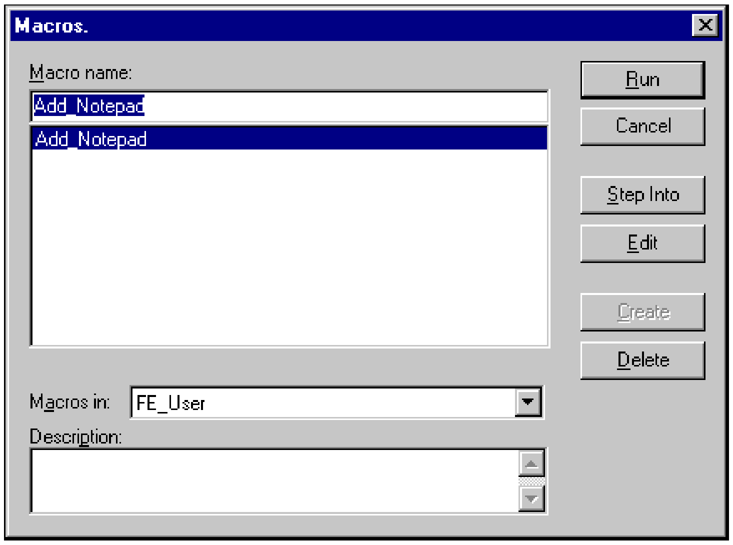
Data Object Macros. With Data Object macros, you can run a set of instructions for the data object you are currently using. All top-level objects and some commonly-used forms have the option to run data object macros. When you run a macro, VBA is called with a data object representing the current record so you can manipulate that record and navigate its object model. You can use Data Object macros to give your users a quick summary of project balances, displaying the information in your custom format.

Query and Export Macros. Query macros provide access to each result row as you process a query. With special VBA user fields, you can modify the query results. For example, you can include a VBA user field that is the sum of two other fields in your query.

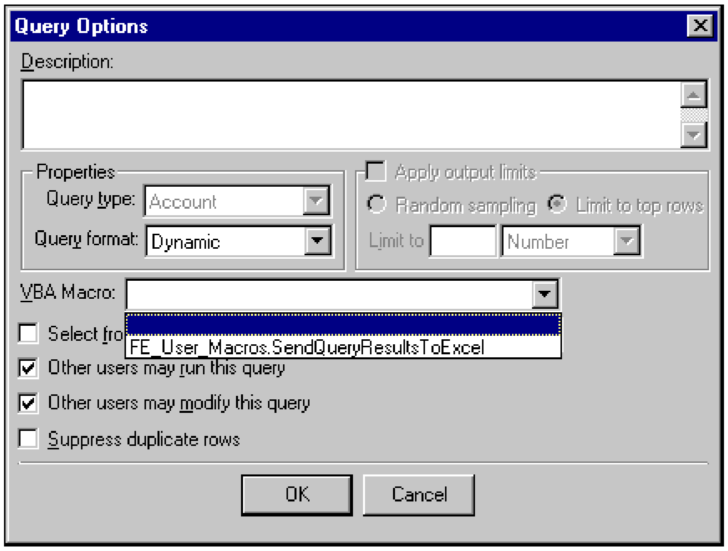
Tips for Debugging and Running Macros

Because VBA is integrated into the program’s shell, you can debug macros while the application is running, regardless of the macro type. You can place breakpoints into the macro code and use all the standard debugging tools provided with the VBA IDE. For specific information about using VBA debugging tools, see the VBA help file accessible from the IDE.

Although you can debug all macro types the same way, you must run each using a different method. After you write an active object macro, it runs automatically when a user takes the action that fires the macro. On the other hand, users must manually run standard macros from the menu bar in ***The Financial Edge*** shell. To run a standard macro, on the ***Financial Edge*** menu bar, select **Tools**, **Run Macro**. The Macros screen appears.


Macros.
Macro name: Add_Notepad
Add_Notepad
Macros in: FE_User
Description:

Then, select a macro and click **Run**. To run a query macro, you must manually add the macro as a query option. To run a query macro from an open query, on the menu bar, select **Tools**, **Query Options**. The Query Options screen appears.



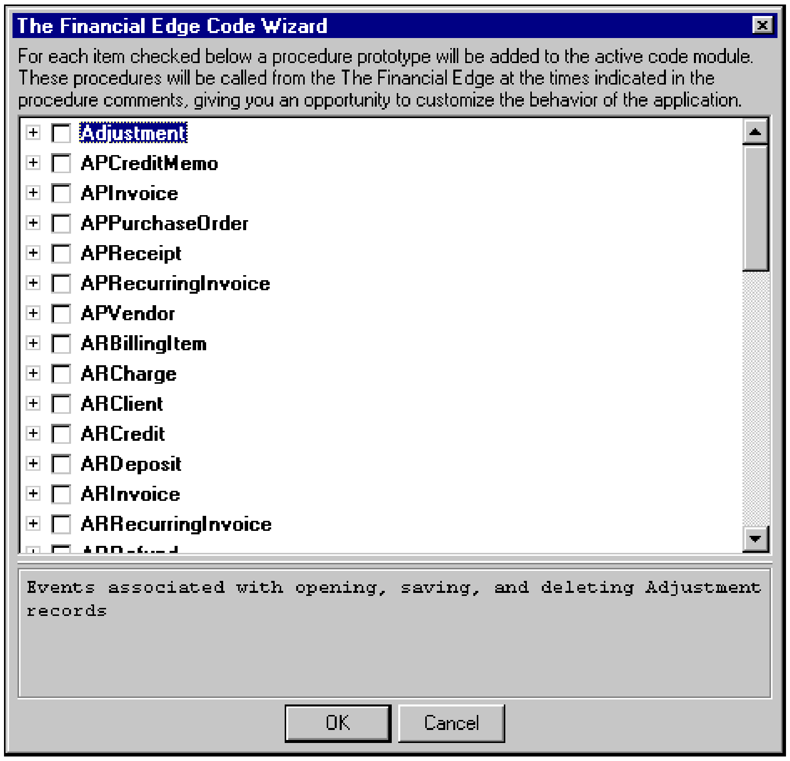
In the **VBA Macro** field, select the query macro and click **OK**. When you run the query, the macro fires once for each row in the result set, again when the process begins, and once more when the process ends.

Managing Active Object Macros

Active object macros execute during certain events in the life cycle of an object or process. For example, you can enter code that executes before accounts are saved. Then, if an account does not meet custom business requirements you specify, you can cancel the save event. The available events vary based on the type of object selected. For more information about specific objects, see “Managing Active Objects” on page 77.

To create active object macros from the VBA IDE, on the toolbar, click **FE Code Wizard**. The Financial Edge Code Wizard screen appears.

Only the Supervisor can create Active Object Macros.



The Financial Edge Code Wizard

For each item checked below a procedure prototype will be added to the active code module. These procedures will be called from the The Financial Edge at the times indicated in the procedure comments, giving you an opportunity to customize the behavior of the application.

- [ ] Adjustment
- [ ] APCreditMemo
- [ ] APInvoice
- [ ] APPurchaseOrder
- [ ] APReceipt
- [ ] APRecurringInvoice
- [ ] APVendor
- [ ] ARBillingItem
- [ ] ARCharge
- [ ] ARClient
- [ ] ARCredit
- [ ] ARDeposit
- [ ] ARInvoice
- [ ] ARRecurringInvoice

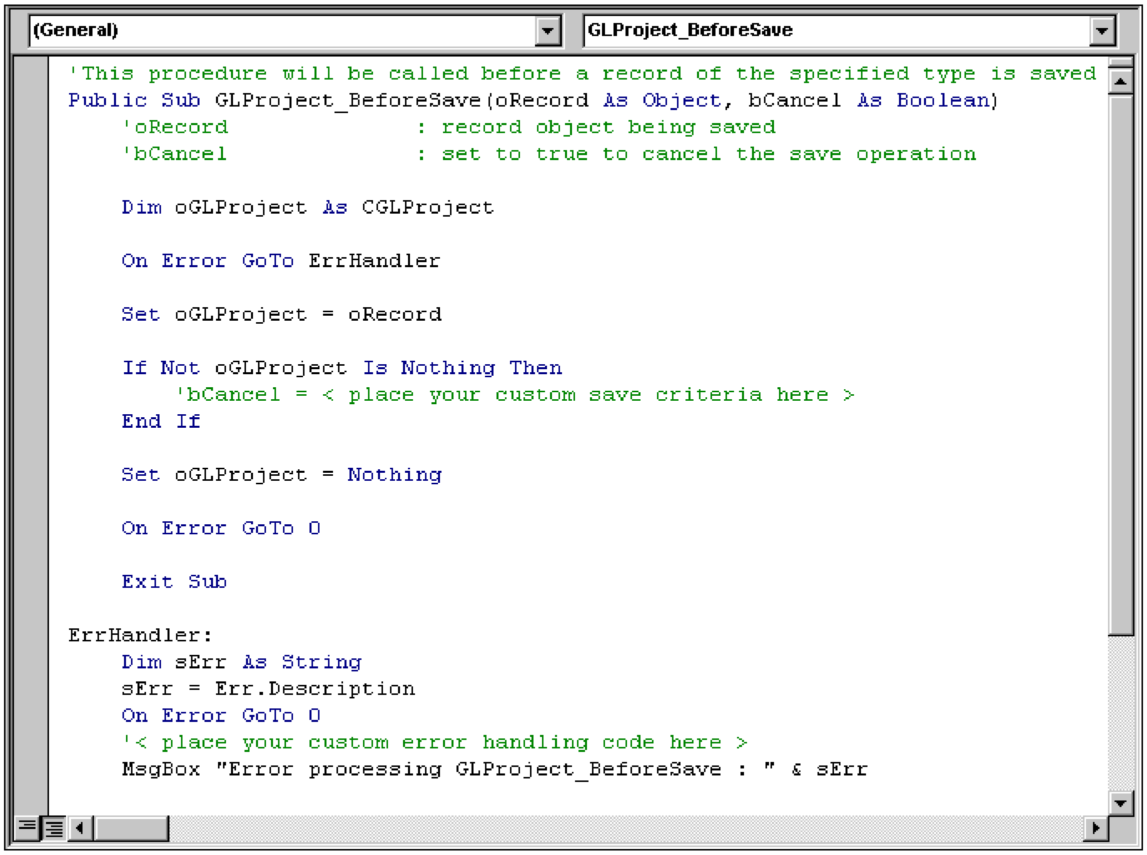
```
Events associated with opening, saving, and deleting Adjustment
records
```

The code wizard simplifies the process of creating active object macros. After you select a set of events you want the macro to respond to, the code wizard generates code that includes useful comments and error checking. For example, you can enforce a rule requiring Project codes that are four digits long.

➢ Creating a custom project business rule

1. From the IDE, click **FE Code Wizard**. The Financial Edge Code Wizard screen appears.
2. From the treeview, mark **GLProject**.
3. Expand the selections under **GLProject** and mark **GLProject_BeforeSave**.

4. Unmark all other project checkboxes and click **OK**. The GLProject_BeforeSave code appears in the code window.



```
'This procedure will be called before a record of the specified type is saved
Public Sub GLProject_BeforeSave(oRecord As Object, bCancel As Boolean)
    'oRecord                : record object being saved
    'bCancel                : set to true to cancel the save operation

    Dim oGLProject As CGLProject

    On Error GoTo ErrHandler

    Set oGLProject = oRecord

    If Not oGLProject Is Nothing Then
        'bCancel = < place your custom save criteria here >
    End If

    Set oGLProject = Nothing

    On Error GoTo 0

    Exit Sub

ErrHandler:
    Dim sErr As String
    sErr = Err.Description
    On Error GoTo 0
    '< place your custom error handling code here >
    MsgBox "Error processing GLProject_BeforeSave : " & sErr
```

In the code, the green code comments explain parameters that are available. oRecord is the project record the user is saving. Even though the record has not been stored in the database, the oRecord object contains all information defined for the unsaved record. oGLProject is the class that manages projects in ***The Financial Edge***. After you set oGLProject to oRecord, you can use Intellisense. For more information about Intellisense, see “Using the Type Library” on page 6.

With the bCancel parameter, you can cancel the save event by adding the following code after the If statement:

```
If Len(oGLProject.Fields(GLPROJECTS_fld_PROJECTID)) <> 4 Then
    MsgBox "Project ID must be four digits"
    bCancel = True
 End If
```

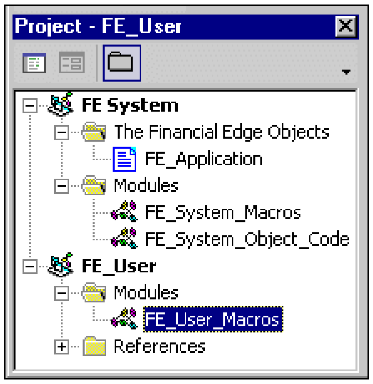
5. After you add your code, save the macro and return to ***The Financial Edge***.
6. To test the macro, enter a project with a five-digit Project ID. A message appears and the save event is canceled.

   For information about debugging and running macros, see “Managing VBA Macros” on page 89.

Managing Standard Macros

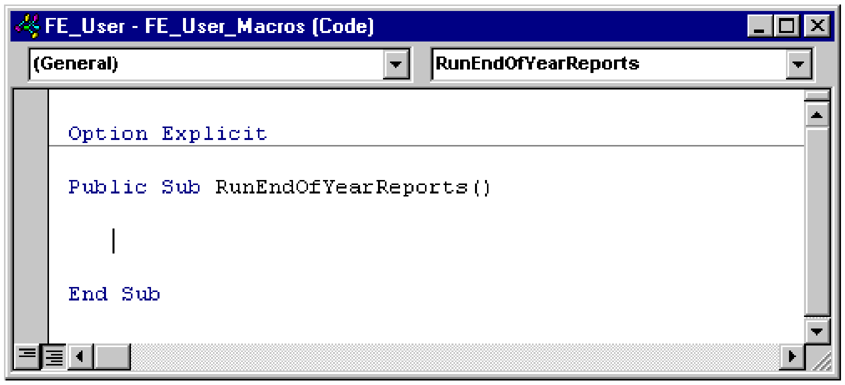
A standard macro is a block of code you can execute from anywhere in ***The Financial Edge*** shell. Creating macros for end users opens the door to a wide range of options that simplify their work and increase efficiency.

To create standard macros from the IDE, in the Project Explorer, select the System or User project. Under the System project is a System_Macros module, and under the User project is a User_Macros module. You can place your macros into these modules or add additional modules as needed.



You can create macros in the System_Macros module in the System project only when you are logged into the accounting system as Supervisor. These macros are available to all users. However, within your macro code you can access the current user name and limit who can actually run the macro. Macros written in the User_Macros module in the User project are available only to the user who created them.

After you access a project module, create a public subroutine in the System_Macros or User_Macros module. Standard macros cannot have any parameters. Keep in mind the name of your macro is displayed to your users, so the name should clearly define the macro's purpose.

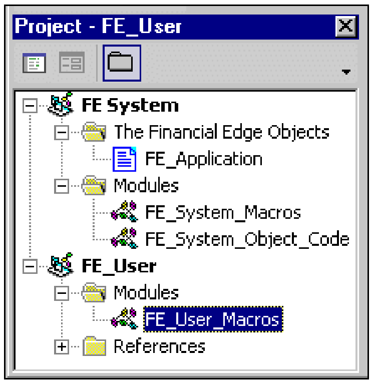


```
Option Explicit

Public Sub RunEndOfYearReports()

End Sub
```

Managing Data Object Macros

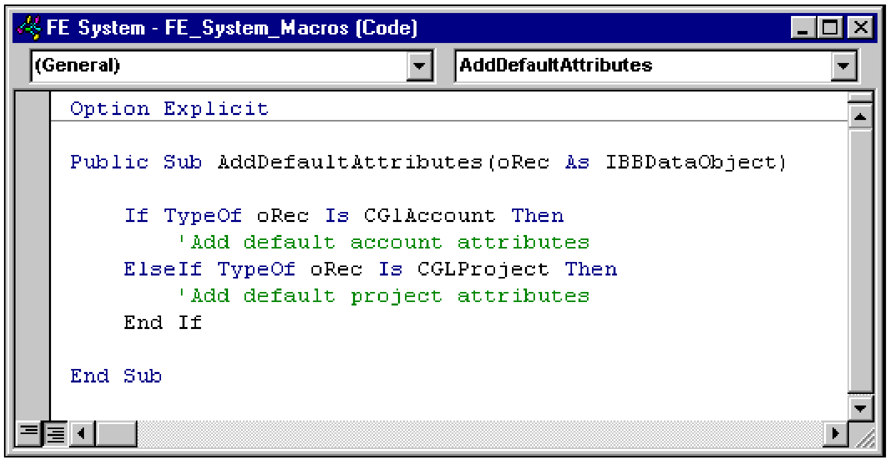
With data object macros, you can run a set of instructions using the data object with which you are currently working. All top-level objects and some other commonly-used forms, such as Notepad and Address, have the option to run a macro using the underlying data object. When the macro is run, VBA is called with the live data object representing the current record, so you can navigate its entire object model. Through data object macros, you can write a calculation to fill in a data field, automatically add a list of attributes to an account, or fill in default values for a record based on user input.

To create data object macros from the IDE, open the Project Explorer. The Project Explorer has two projects — System and User. Under the System project is a System_Macros module, and under the User project is a User_Macros module. You must place your macros into these modules for them to be recognized by ***The Financial Edge*** as Data Object macros. You can add additional modules as needed to provide support for your System or User macros.



You can create macros in the System_Macros module in the System project only when you are logged into the accounting system as Supervisor. These macros are available to all users. However, within your macro code you can access the current user name and limit who can actually run the macro. Macros written in the User_Macros module in the User project are available only to the user who created them.

After you access a project module, create a public subroutine in the System_Macros or User_Macros module. Data Object macros must have one parameter of the *IBBDataObject* type. Without this parameter, the program will not recognize your macro as a Data Object macro.



```
Option Explicit

Public Sub AddDefaultAttributes(oRec As IBBDataObject)

    If TypeOf oRec Is CGlAccount Then
        'Add default account attributes
    ElseIf TypeOf oRec Is CGLProject Then
        'Add default project attributes
    End If

End Sub
```

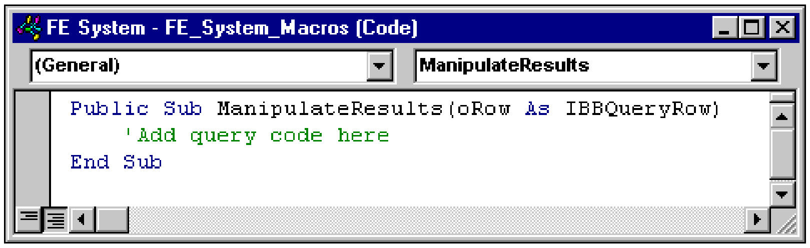
Data Object macros are available from any data object supporting a data object macro. For example, even though a macro is designed for the project data object, it will still be available when the user looks at the macro list from an account record. Therefore, you need to make sure that the data object being passed in is of the correct type for that particular macro. For a code sample that shows you how to check the data type of a passed-in data object, see "Sample Data Object Macro: Setting Defaults" on page 97.

Managing Query Macros

Using Query macros, you can detect the start and end of the query process and modify the results through the VBA User fields. Once written, the macros are attached to the query.

To create query macros from the IDE, in the Project Explorer, select the System or User. Under the System project is a System_Macros module, and under the User project is a User_Macros module. Query macros must be written in one of these modules to be recognized.

After you access a project module, create a public subroutine in the System_Macros or User_Macros module. Query macros must have one parameter, IBBQueryRow. Without this parameter, the program does not recognize your macro as a query macro.



```
Public Sub ManipulateResults(oRow As IBBQueryRow)
    'Add query code here
End Sub
```

Using the VBA User Field

In *Query*, one **VBA User Field** appears in the **Available Fields** list for each query type. If you select this field as an output field, you can use it as a read/write field and can manipulate it with a VBA Macro. This is the only field in the query results to which the user has write access during query processing. *Query* does not put anything in this field — for this field to be populated or used, you must use a VBA Macro. Note that the **VBA User Field** appears only in the **Available Fields** list once per query type. However, you can select it more than once and follow standard naming conventions to define additional user fields in a query.

The following query macro shows an example using BOF, EOF, and the **VBA User Field** to analyze expense accounts and compare each account's activity to its corresponding budget by calculating a difference in the **VBA User Field**. The one parameter for the macro must be of the IBBQueryRow type. The query includes the **Account Number**, **Net Activity**, **Annual Budget**, and **VBA User Field** output fields.

```
Public Sub UserQueryMacro(oQueryRow As IBBQueryRow)

    Const ACCOUNT_NUMBER = 1
    Const NET_ACTIVITY = 2
    Const ANNUAL_BUDGET = 3
    Const VBA_FIELD = 4

    If oQueryRow.BOF Then
        'This code is processed once (at the beginning of
        'the query result set).
        MsgBox "Begin processing"
    ElseIf oQueryRow.EOF Then
        'This code is processed once (at the end of the
        'query result set).
        MsgBox "End processing"
    Else
        'This code is processed once per row
        'You can only write to a "VBA User field"
         oQueryRow.Field(VBA_FIELD) = ANNUAL_BUDGET - NET_ACTIVITY
    End If

End Sub
```

When you run the query:

1. A “Begin processing” message appears. To continue processing the query, click **OK**.

2. For each row, the net activity of the account is subtracted from the budget to generate the difference in the **VBA User Field**.
3. An “End processing” message appears. To close the message, click **OK**.

Note that when viewing the query results in the Results window, the EOF is not triggered unless the user scrolls to the end of the result set or closes the query. To run the macro, from a query, select **Tools**, **Run Macro**.

Macro Samples

This section contains three sample macros that demonstrate the basic concepts of writing macros. The first, “Setting Defaults”, is a data object macro that sets a default start date for project records. The second, “Adding Notepad Records”, is a standard macro that uses several VBA objects and features to prompt users for an account and display a Notepad user interface for adding notepad records. The third example, “Exporting to Excel”, is a query macro that exports query results to *Excel*. For more code samples, see “Sample Programs” on page 123.

Sample Data Object Macro: Setting Defaults

This is a simple macro for adding default information to project records, but you can easily adapt this code for any record type. It demonstrates two important ideas. First, the data object macro is available from several different data entry forms, so it is important to check the data object type before using any of its properties. Second, you have access to the data object and all its child collections and classes, so that you can add, edit, and delete information when necessary.

```
Public Sub SetDefaults(oRecord As IBBDataObject)

    'Check the record type
    If TypeOf oRecord Is CGLProject Then

        Dim oProject As CGLProject

        'Setting the oProject = oRecord is not necessary
        'but it activates IntelliSense for the object
        Set oProject = oRecord

        oProject.Fields(GLPROJECTS_fld_STARTDATE) = FormatDateTime(Now, vbShortDate)

    ElseIf TypeOf oRec Is CGlAccount Then

        'Add default account information

    End If

End Sub
```

Sample Standard Macro: Adding Notepad Records

This sample uses many *VBA* features. With this macro, users can add notepads to an account without first loading the account.

```
Public Sub Add_Notepad()

    Dim lID As Long

    Dim oAccount as CGLAccount
    Dim FEService as FE_Services

    Dim oQuickSearch As IBBMiscUI
    Dim oNotepadForm As CNotepadForm

    Set FEService = New FE_Services
    FEService.Init FE_Application.SessionContext

    'Search for the account using the Quick Search form
    Set oQuickSearch = FEService.CreateServiceObject(bbsoMiscUI)
    oQuickSearch.Init FE_Application.SessionContext

    lID = oQuickSearch.PromptForDataObject(SEARCH_GLACCOUNT, _
                    "Account", _
                    "Search for an account")

    'If an account is found, then open a notepad form
    If lID > 0 Then

        Set oAccount = New CGLAccount
        oAccount.Init FEApplication.SessionContext

        Set oNotepadForm = New CNotepadForm
        oNotepadForm.Init FEApplication.SessionContext

        oAccount.Load lID

        With oNotepadForm
            Set .NotepadObjects = oAccount.Notepads
            .ShowForm Nothing, , True, False
        End With

        oAccount.Save
    Else
        MsgBox "Account not found"
    End If

    'Clean up
    If Not oAccount Is Nothing Then
        oAccount.CloseDown
        Set oAccount = Nothing
    End If
```

(Continued- page 2 of 2)

```
    If Not oNotepadForm Is Nothing Then
        oNotepadForm.CloseDown
        Set oNotepadForm = Nothing
    End If

    oQuickSearch.CloseDown
    Set oQuickSearch = Nothing

    FEService.CloseDown
    Set FEService = Nothing

End Sub
```

Sample Query Macro: Exporting to Excel

For additional properties you may find helpful when moving through query results, see IBBQueryRow in the Object Reference section of the VBA and API help file.

```
'This sample requires a reference to Excel Object Library.
Option Explicit

Private moExcel As Excel.Application
Private moWorksheet As Excel.Worksheet

Public Sub SendQueryResultsToExcel(oRow As IBBQueryRow)

    If oRow.BOF Then
        'Opens Excel
        Set moExcel = Excel.Application
        moExcel.Visible = True

        'Add a new worksheet
        moExcel.Workbooks.Add
        Set moWorksheet = moExcel.Worksheets(1) '.Add

        'Fills the first row with the field names from the query
        Dim lHeads As Long
        For lHeads = 1 To oRow.FieldCount - 1
            moWorksheet.Cells(1, lHeads) = oRow.FieldName(lHeads)
        Next lHeads

    ElseIf oRow.EOF Then
        ' Post-process some results in Excel
        moWorksheet.Columns("A:E").EntireColumn.AutoFit

        'Clean up
        Set moWorksheet = Nothing
        Set moExcel = Nothing

    Else

        ' Fill In The Details
        Dim l As Long
        For l = 1 To oRow.FieldCount - 1
            'Uses the data from the query to move to Excel
            moWorksheet.Cells(oRow.RowNum + 1, l) = oRow.Field(l)
        Next l

    End If
```

Chapter 4

Blackbaud API

Contents

This chapter introduces *API*. With this optional module, you can leverage the power of ***The Financial Edge*** programmatically from third party or custom applications. You can use any program that can manipulate *Visual Basic* COM objects to write stand-alone applications for *API*. You can also integrate programs that support VBA, such as Microsoft *Office*, with ***The Financial Edge***. For example, you can use ***Financial Edge*** account activity to produce pivot tables or graphs in Microsoft *Excel*. To speed code entry, you can reuse many of the native data entry forms and search screens in your API application.

To customize your software, you can “extend” the ***Financial Edge*** shell by building plug-ins or add-ins your users can run while they are in the program. For example, you can create a plug-in to run several queries, export the data to Microsoft *Excel*, and produce reports that include charts and graphs. Or you can create a plug-in with a data entry form for adding information to the database in a custom format. After you create a plug-in, you can add it to the Plug-Ins page of the program shell to access it quickly. To access plug-ins from the Plug-Ins page, on the navigation bar, click **Plug-Ins**, then click the link for the plug-in you want to run.

To further improve accessibility to your accounting data, you can use *API* to access your data through the Internet using tools available in Microsoft’s *Internet Explorer*. The *Windows Scripting Host* provided with the *Windows* 32-bit operating systems enables you to use scripts that are similar to batch files to access your data.

Working with the API

To successfully integrate API applications with ***The Financial Edge***, you should include three critical features in your programming. The first is a reference to ***Financial Edge*** type libraries so you can gain early-bound access to ***Financial Edge*** objects. Second, you should follow proper declaration, startup, terminate, and closedown methods. Finally, to create a successful integration, you must successfully access the program database. The following sections explain common API code conventions and methods. For procedures on referencing ***Financial Edge*** type libraries, see “Using the Type Library” on page 6.

API Code Conventions

To connect with ***The Financial Edge*** through the API, you must follow certain code conventions in your applications. For example, in the declarations section of your project, you must create a global object reference to the FE_API object. This reference should be global because it is created and initialized only once when you start the program. You can then use the object in various places throughout the program to manipulate other objects, but you should destroy the object when the program terminates to prevent memory leaks.

For brevity, common code segments such as the declaration, startup, terminate, and closedown statements are omitted in later code samples in this chapter, however, your applications cannot function without them.

The following code samples illustrate declaration, startup, terminate, and closedown methods in API:

```
'Place this in the declarations section of your program
Public goFE_API as FE_API
'Place this in the startup section of your program
Set goFE_API = New FE_API
'This forces an 'Exit and Sign Out' when your goFE_API object is destroyed
goFE_API.SignOutOnTerminate = True
'Place this in the closedown section of your program
Set goFE_API = Nothing
```

Accessing the API

Regardless of the method you use to access ***The Financial Edge***, to access the API, you must call the FE_API.Init method just after the application startup. The FE_API.Init method requires that you provide up to four pieces of information to establish a connection to the database, depending on the method you use. This information includes:

- User Name. You assign user names through the Set up system security link in *Administration*.

- Password. You can assign passwords to new users.
- Database Number. If you have multiple databases installed, each has an assigned database number. For example, the ***Financial Edge*** sample database is assigned number 50.
- Serial Number. You must always supply the serial number, or at least the serial number parameter using double quotation marks (“ ”). The other three parameters are optional when calling the FE_API.Init method.

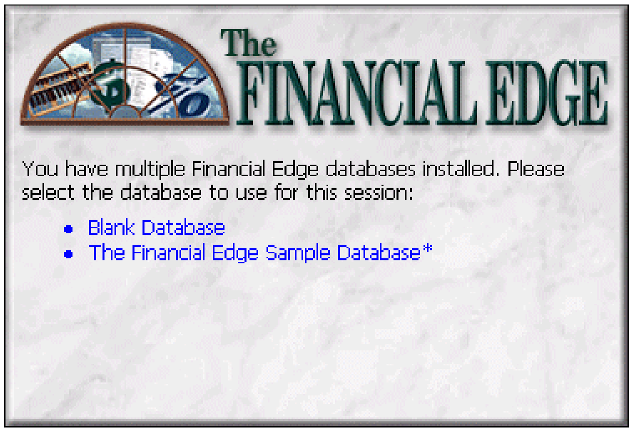
You can use one of three methods to connect to a ***Financial Edge*** database, unless you are a member of the Blackbaud Developer Network (BDN) for third party vendors. If you are a member of the BDN, you must use Option 4, which involves passing the serial number and sThirdParty arguments you received from Blackbaud when you purchased *API*.

Option 1: Using the Financial Edge Login Form

The easiest method for connecting to ***The Financial Edge*** is to use the standard login screen and specify only the database serial number.

```
bLoginOK = goFE_API.Init(SERIAL_NUMBER)
```

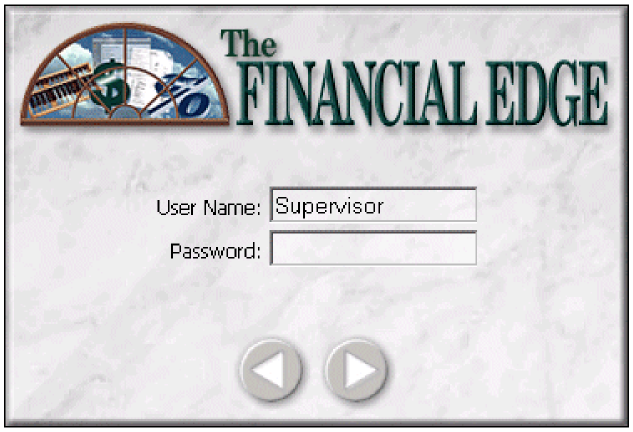
In this case, because you do not identify a specific database number, the initialization code determines whether you have multiple databases installed. If so, a screen appears asking you to select a database:



You have multiple Financial Edge databases installed. Please select the database to use for this session:

- Blank Database
- The Financial Edge Sample Database*

After the user selects a database, or if only one database is installed, the program asks for a user name and password if you did not specify one. To completely bypass the login screen, you can add all three parameters to the login code.



User Name: Supervisor
Password:

Option 2: Bypassing the Login Form

If you have all required information, you can connect directly to a ***Financial Edge*** database and bypass the login forms. In this case, assuming all information is valid, you connect directly to the database without having to enter information on additional screens. This method is not ideal because it makes the user password accessible to anyone with access to your program code. However, it may be useful in cases where you need an application to connect to the database without user input. For example, you may want an employee to launch an application before leaving work, so the program can extract information from the database at night. Using this method, you can permit anyone to run the application with supervisor rights, without giving out your password.

```
bLoginOK = goFE_API.Init("", "Supervisor", "Admin", 50)
```

Option 3: Creating A Custom Login Form

This option is essentially the same as Option 2, but you create a custom user interface that asks users to select a database and provide their user name and password. The FE_API object provides the GetAvailableRegistryKeys method, which returns a list of the registry keys for each ***Financial Edge*** database you have installed. With this information, you can extract additional information from the registry, including the database description, the DSN, and the System Directory, where ***Financial Edge*** program files are located.

```
bLoginOK = goFE_API.Init("", sUserName, sPassword, lDatabaseNumber)
```

Option 4: Accessing The Financial Edge for Third Party Vendors

Third party vendors must always include the SerialNumber and sThirdPartyVendor arguments supplied by Blackbaud at time of purchase.

API and VB.NET

Coding API applications in *VB.NET* requires special consideration. When using *API* with *VB.NET*, you should:

1. Remember to reference the API. To make a reference to the API, from the menu bar, select **Project**, **Add Reference**. On the COM tab, select **Blackbaud FE7.0 Objects**, then click **OK**.
2. Load a project form before loading other forms. This means ***The Financial Edge*** splash screen or login form should not appear before one of the project forms. Because of this restriction, you should not put the code to initialize the API in the Load event of your form. Instead, use one of the following methods:
   - Display a splash screen first and write the code to initialize the API in the Load event of the main form.
   - Start a timer with a small time interval in the Load event of your form. Initialize the API only when the timer goes off.
   - Run ***The Financial Edge*** as the server by specifying the optional parameter lAppmode in the initialization. You must correctly specify all parameters to run *API* in this mode.
3. Intellisense for ***The Financial Edge*** works in *VB.NET*. In cases where it does not work, remember that Enum members should be preceded by the name of the Enum. For example, in *Visual Basic 6.0*, to set the description field of an object oProject of type CGLProject, you would use the following code:

   ```
   oProject.Fields(GLPROJECTS_fld_DESCRIPTION) = "This is just a demo"
   ```

   In *VB.NET*, you should use this code:

   ```
   oProject.Fields(EGLPROJECTSFields.GLPROJECTS_fld_DESCRIPTION) = "This is just a demo"
   ```

Some functions of *API*, such as the LoadProductCombo of the IBBUtilityCode, are specific to *Visual Basic 6.0*. This function takes a ComboBox argument and loads the product names into it. This and similar functions do not work in *VB.NET*, so you must write your own functions to do the work. For more details, see the LoadCombo function in the .NET Sample.

For a code sample for creating a CGLProject with *VB.NET*, see the CProjectSetup.cls file in the Help\Samples\Advanced_Samples\API\Samples\GL folder of the installation directory.

Managing the FE_API Object

The FE_API object represents the entire ***Financial Edge*** program in *API*, and it provides access to a valid SessionContext for initializing objects. It is important that you maintain a reference to the FE_API object throughout the lifetime of your API program because, when this object is released, your connection to the accounting system is closed. For introductory information about the SessionContext for API, see “Initializing and Releasing Objects” on page 14.

The SessionContext Property

The SessionContext is the most popular method used in ***The Financial Edge***, although it is not used during the initial log in. The SessionContext holds information about the state of the active instance of the application, and it is required to initialize all other objects in the system. Each time you create a top-level object, it must be initialized with a valid SessionContext.

This code sample shows how to initialize an account object.

```
'Create a new Account object and initialize it with our current SessionContext
Set oAccount = New CGLAccount
oAccount.Init moFE_API.SessionContext
```

The AppMode Property

AppMode is a read-only property used to determine whether the application is running standalone or as part of a server, such as a Web server.

```
'Are we running standalone?
If moFE_API.AppMode = amStandalone Then
    'Do standalone code
Else
    'Do server code
End If
```

The GetAvailableRegistryKeys Method

The GetAvailableRegistryKeys method returns an array containing the registry key root for each installed ***Financial Edge*** database. If you have multiple ***Financial Edge*** databases installed with API support, you can use this method to present users with a list of available databases. The FE_API object does not have to be initialized to access this method.

```
'Check the registry root of the first installed Financial Edge database
Debug.Print moFE_API.GetAvailableRegistryKeys(1)
```

The following code illustrates using the GetAvailableRegistryKeys method to populate a combo box with a list of available ***Financial Edge*** databases:

```
'Enter 'Private moFE_API as FE_API' in the general declarations section

Private Sub UserForm_Initialize()

    Dim lCntr As Long
    Dim vDatabases As Variant

    'Create an instance of the Financial Edge API
    Set moFE_API = New FE_API

    'Get a list of available FE databases
    vDatabases = moFE_API.GetAvailableRegistryKeys

    'Load a combo with the available choices
    With cboDatabases
        .Clear
        For lCntr = 1 To UBound(vDatabases)
            .AddItem vDatabases(lCntr)
        Next lCntr
    End With

End Sub
```

Once you have access to the registry key, you can extract descriptive information from the registry to display for users. Registry keys are returned as \Software\Blackbaud\AFNINI_##, where ## is the database ID number.

Three useful keys are:

Key	**Sample**
<*key*>\GENERAL\DB_DESCRIPTION	“The Financial Edge Sample Database”
<*key*>\GENERAL\DSN	\HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\AFNINI_50
<*key*>\GENERAL\SYSTEMPATH	C:\Program Files\The Financial Edge

The LastErrorMessage Property

LastErrorMessage is a read-only property you can use to display the reason for an Init method failure.

```
'Initialize the FE_API object and attempt to connect to the FE sample database
If Not moFE_API.Init("", "Supervisor", "Admin", 50) Then
    MsgBox "Cannot connect to database for the following reason: " _
      moFE_API.LastErrorMessage, vbOKOnly Or vbInformation
Exit Sub
End If
```

The QueryShutDown Method

You can place the QueryShutDown method in the QueryUnload event of your main form to ensure that all non-modal ***Financial Edge*** forms are unloaded. QueryShutDown returns False if one or more of the non-modal forms cannot be unloaded.

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    'Make sure all non-modal FE_ forms are closed
    If Not moFE_API.QueryShutDown Then
        Cancel = True
    End If
End Sub
```

The SignOutOnTerminate Property

If set to True (-1), this property forces your instance of ***The Financial Edge*** to log off when the FE_API object is released.

```
'This causes the application to log off The Financial Edge.
'If the property is not set to "True", the user remains logged into
'The Financial Edge when the custom code ends.
moFE_API.SignOutOnTerminate = True
Set moFE_API = Nothing
```

Managing the FE_Services Object

The FE_Services object provides access to a number of common forms, objects, and collections through the CreateServiceObjects method. For more information about bbServiceObjects, see the “Object Reference” section of the VBA and API help file.

The CreateServiceObject Method

FE_Services exposes a number of commonly used forms, objects, and collections through the CreateServiceObject method. This example uses CreateServiceObject to create an IBBReportInstances collection object that gives you access to all existing income statement reports.

The following code samples illustrate creating a reference, creating an instance, and initializing the IBBReportInstances collection.

```
'Create a reference to an IBBReportInstances collection
Dim oReports As IBBReportInstances

'Create an instance of the IBBReportInstances collection
Set oReports = oFE_Services.CreateServiceObject(bbsoReportInstances)

'Initialize the collection with the SessionContext and the
'report type for an Income Statement
oReports.Init oFE_API.SessionContext, bbrep_GL_IncomeStatement
```

For a list of service objects you can create, see the “Object Reference” section of the VBA and API help file.

The GetProgID Methods

You can use three methods of the FE_Services object to return the full class name of objects that exist in *API*. With the class name, you can use the CreateObject method that is part of *Visual Basic* to create an instance of the class using the application name and class ID.

```
'Add these variable declarations
Dim oRecord As Object
Dim sClassName As String

'Get the class name for the Account data object
sClassName = oFE_Services.GetProgIDForDataObject(bbdataGLAccount)

'Create the data object by the Class Name
Set oRecord = CreateObject(sClassName)
```

```
'Add these variable declarations
```

Syntax for the full class name is AppName.ObjectType, for example, GLAccountData7.CGlAccount. The three methods for returning full class names are:

- GetProgIDForDataObject accepts a member of the bbDataObjConstants enum as a parameter and returns the full class name for a data object.
- GetProgIDForUIObject accepts a member of the bbDataObjConstants enum as a parameter and returns the full class name for a user interface object.
- GetProgIDForMetaProvider accepts a member of the bbMetaObjects enum as a parameter and returns the full class name for a meta object.

For more information about GetProgIDForDataObject and GetProgIDForUIObject, see the bbDataObjConstants topic in the Object Reference section of the VBA and API help file. For more information about GetProgIDForMetaProvider, see the bbMetaObjects topic.

Managing Plug-Ins

Plug-ins are specially built applet extensions of ***The Financial Edge***. A plug-in does just that, it “plugs in” to the program user interface, opening the door to a wide range of custom functionality. Plug-ins can be as simple as an HTML page or Microsoft *Office* document, or as complicated as a multi-level ActiveX document or interactive spreadsheet. With the flexibility of plug-ins, you can add custom applications and extensions directly into the accounting package via the Plug-Ins page, which is accessible from the navigation bar. Plug-ins share the database connection and runtime code resources with ***The Financial Edge***, making them an excellent choice for adding custom functionality without the overhead of having to build a full-blown API application. In order to build a plug-in, you must first build a special COM dynamic link library (DLL) using *API*.

The typical plug-in is made of two parts. The first is a class module that implements the IBBHostedPlugin and the IBBHeaderInfo interface, and which provides information about the plug-in. The second is a document that provides the user interface. The document type depends on the application you want to build.

Part One: The IBBHostedPlugin and IBBHeaderInfo Interface

The IBBHostedPlugin and IBBHeaderInfo interface classes provide information to the host, such as the name and description of the plug-in. You should place these classes in the General Declarations section of your plug-in and fill in each of the properties and events.

The following table displays properties of the IBBHostedPlugin class:

Property	Description
ProgID	Specifies the name of the document to load.
URL	Complete path to document specified above. Must include document name. (Use App.Path if document is in same directory as the DLL.)

With the four IBBHostedPlugin events, you can respond to system and user actions. The following table describes the four IBBHostedPlugin events:

Event	Description
OnInit	Occurs before all other events when the host creates an instance of the plug-in object.
OnLoad	Occurs when the plug-in document is loaded into the shell and before control is passed to the user.
OnQueryUnload	Occurs before the document is unloaded.
OnClosedown	Occurs when the host destroys the document.

The following table displays properties of the IBBHeaderInfo class:

Property	Description
Name	This is the text displayed in the right **Plug-In** column of the Plug-ins page.
Description	This is the text displayed in the left ’Description’ column of the Plug-Ins page.
Caption	Appears in the top frame when the plug-in is loaded.
Image	Lets you specify a graphic to display to the left of the header caption.
Group	Reserved for future use. No need to enter anything.

Part Two: The User Interface

You can specify a wide array of document types in the ProgID and URL properties, listing ***The Financial Edge*** as the host. For example, to host an *Excel* spreadsheet, you would use the following:

```
Private Property Get IBBHostedPlugin_ProgID() As String
    IBBHostedPlugin_ProgID = "pMyPlugin.xls"
End Property

Private Property Get IBBHostedPlugin_URL() As String
    IBBHostedPlugin_URL = App.Path & "\docMyActiveXPlugin.xls"
End Property
```

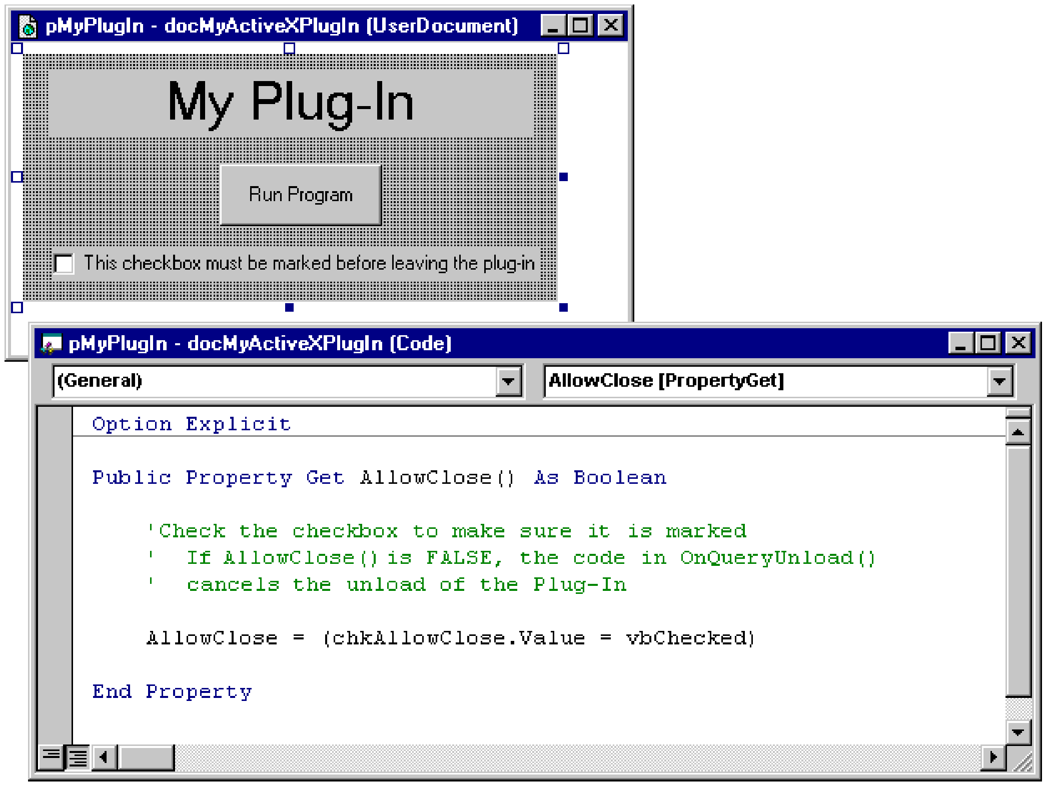
To host a local HTML page:

```
Private Property Get IBBHostedPlugin_ProgID() As String
    IBBHostedPlugin_ProgID = "pMyPlugin.html"
End Property

Private Property Get IBBHostedPlugin_URL() As String
    IBBHostedPlugin_URL = App.Path & "\docMyActiveXPlugin.html"
End Property
```

To host an ActiveX user document:

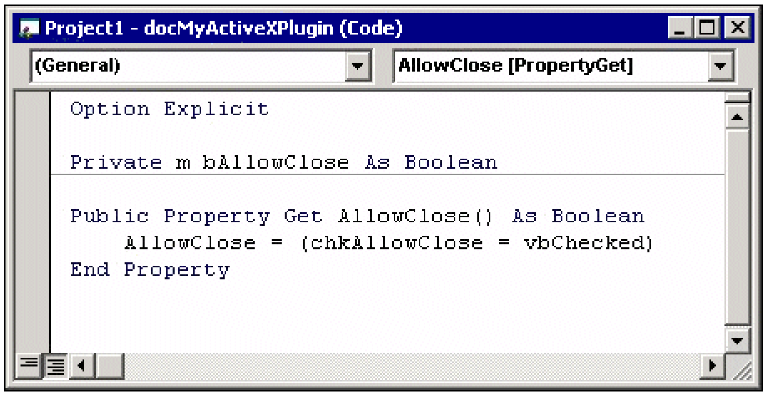
```
Private Property Get IBBHostedPlugin_ProgID() As String
    IBBHostedPlugin_ProgID = "pMyPlugin.vbd"
End Property

Private Property Get IBBHostedPlugin_URL() As String
    IBBHostedPlugin_URL = App.Path & "\docMyActiveXPlugin.vbd"
End Property
```

You can then create a UserDocument to provide a user interface.


pMyPlugIn - docMyActiveXPlugIn (UserDocument)
My Plug-In
Run Program
This checkbox must be marked before leaving the plug-in
pMyPlugIn - docMyActiveXPlugIn (Code)
(General)
AllowClose [PropertyGet]

```
Option Explicit

Public Property Get AllowClose() As Boolean

    'Check the checkbox to make sure it is marked
    '   If AllowClose() is FALSE, the code in OnQueryUnload()
    '   cancels the unload of the Plug-In

    AllowClose = (chkAllowClose.Value = vbChecked)

End Property
```

In this example the UserDocument contains a single property, AllowClose, which fires when the cMyActiveXPlugIn.cls plug-in calls the OnQueryUnload event. The AllowClose property prevents the user from closing the plug-in without marking the checkbox at the bottom of the form. This method is useful for validating user input or ensuring that all required tasks are completed before closing the plug-in.

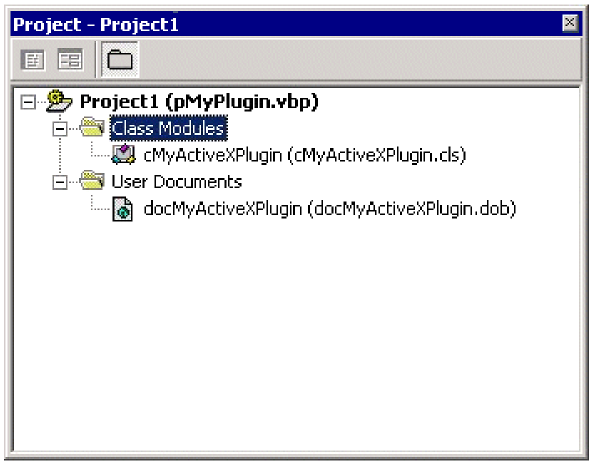


```
Option Explicit

Private m_bAllowClose As Boolean

Public Property Get AllowClose() As Boolean
    AllowClose = (chkAllowClose = vbChecked)
End Property
```

Creating Plug-Ins

To create a plug-in, you must first set references to the Blackbaud FE 7.0 object library, the Blackbaud Common Shell Interfaces 7.0 type library, and any other libraries you need. For more information about referencing a type library, see “Using the Type Library” on page 6. Then, you create a new ActiveX DLL project, add a class (or rename the default “Class1”, if it appears), and add a UserDocument module. The class modules serve as a central location for you to store all your plug-ins, so you can reuse common forms and code. A single DLL can contain any number of class modules that implement the IBBHostedPlugin and the IBBHeaderInfo interface. After you set up your classes, you should add references to the IBBHostedPlugin and IBBHeader interface.

This picture shows a class module:


Project - Project1
Project1 (pMyPlugin.vbp)
Class Modules
cMyActiveXPlugin (cMyActiveXPlugin.cls)
User Documents
docMyActiveXPlugin (docMyActiveXPlugin.dob)

➢ Setting up a simple plug-in class

This procedure and sample code use *Visual Basic 6.0* to create a COM dynamic link library (DLL) called *pMyPlugIn.DLL*.

1. To set up a plug-in class, set references to the Blackbaud FE 7.0 object library and the Blackbaud Common Shell Interfaces 7.0 type library. If you need to reference other libraries, you should also set those references.

2. Add a reference to the interface.

```
Implements IBBHostedPlugIn
Implements IBBHeaderInfo
'A session context for the application is passed in when the
'plug-in is initialized (IBBHostedPlugIN_OnInit()). This may not
'be necessary, depending on the application.
Private moSessionContext As IBBSessionContext

'When the plug-in is loaded (IBBHostedPlugIn_OnLoad()), the main
'user interface document is passed in. This may not be
'necessary, depending on the application.
Private moUserDoc As docMyActiveXPlugin ' use if ActiveX Page
```

3. Set initialization information. Note that when a plug-in is first accessed, ***The Financial Edge*** host must perform several initialization tasks before the plug-in is loaded. Because these tasks require that the OnInit and OnCloseDown events fire, you cannot use those events to determine whether or not the plug-in has been run. You should avoid putting code into these events, with the exception of setting a reference to the SessionContext in OnInit and clearing the reference in OnCloseDown. OnLoad and OnQueryUnLoad do not fire during the initialization process, so you can use them for any required start-up and closedown code.

```
Private Sub IBBHostedPlugin_OnInit(ByVal oApp As IBBShellInterfaces.IBBHostedApp)
    Set moContext = oApp.System.SessionContext
End Sub

Private Sub IBBHostedPlugin_OnLoad(ByVal oDoc As Object)
    Set moUserDoc = oDoc
End Sub
```

4. Specify the name of the plug-in user interface document. The IBBHostedPlugin_URL property should return the path to the user interface file. If your interface consists of user documents, when you create the DLL, the user documents are created in the same directory as the *.vbd DLL.

```
'Uses App.Path to return the path to your compiled DLL file.
Private Property Get IBBHostedPlugin_ProgID() As String
    IBBHostedPlugin_ProgID = "pMyPlugin." & TypeName(moUserDoc)
End Property

Private Property Get IBBHostedPlugin_URL() As String
    IBBHostedPlugin_URL = App.Path & "\docMyActiveXPlugin.vbd"
End Property
```

5. Create a user-friendly description for your plug-in. This text appears as a link on the Plug-ins page.

```
Private Property Get IBBHeaderInfo_Name() As String
    IBBHeaderInfo_Name = "My ActiveX Plug-in"
End Property
Private Property Get IBBHeaderInfo_Description() As String
    IBBHeaderInfo_Description = "A Simple Example of an ActiveX Plug-In"
End Property
```

6. Create a caption for the plug-in and add a graphic, if desired. These appear at the top of the Plug-Ins page. Header images can be *.jpg, *.gif, or *.bmp format, but they are restricted to 32 x 32 pixels.

```
Private Property Get IBBHeaderInfo_Caption() As String
    IBBHeaderInfo_Caption = "My ActiveX Plug-in"
End Property

Private Property Get IBBHeaderInfo_Image() As String
        IBBHeaderInfo_Image = App.Path & ?\MyPlugIn.jpg?
End Property
```

7. Close down the plug-in properly. The IBBHostedPlugIn_OnQueryUnload occurs before either the plug-in or application closes. Linking to a separate HTML page from the shell or switching to another shell menu item also fires this event. With OnQueryUnload, you can verify information and cancel the close process if the user did not complete all necessary tasks. If you set bCancel to true, it cancels the unload and returns the user to the plug-in form. If the user tries to close ***The Financial Edge***, bShellIsUnloading returns True.

```
Private Sub IBBHostedPlugin_OnQueryUnload(bCancel As Boolean, ByVal bHostIsUnloading
As Boolean
    'AllowClose is a public method on the docMyPlugIn user
    'document. The routine validates the user input
    'and determines if the plug-in can be closed.
    If Not moUserDoc.AllowClose Then
        MsgBox "Checkbox must be marked in order to close this plug-in."
        bCancel = True
    End If
End Sub

Private Sub IBBHostedPlugin_OnClosedown()

    'Clean up module level variables
    If Not moUserDoc Is Nothing Then
        Set moUserDoc = Nothing
    End If

    If Not moContext Is Nothing Then
        Set moContext = Nothing
    End If
End Sub
```

Deploying Plug-Ins

After you compile your plug-in, you must place a copy of the plug-in DLL and any relevant support files in the *Financial Edge\Plugins* directory. After the files are in place, you can open the plug-in using the link on the Plug-ins page.

Managing API Applications

The flexibility of *API* gives you endless resources for customizing or enhancing ***The Financial Edge***. You can extend the accounting program's inherent functionality, access your database from third-party software, add plug-in features, or use the ***Financial Edge*** architecture to create your own applications. The following sections include code samples illustrating some basic functions you can perform with *API*. For additional sample API programs, see the Financial Edge\Help\Samples directory.

Sample: Adding an Annotation Form

The Annotation service object is a child object, meaning it requires a parent record that has been initialized and loaded. When the Annotation form appears, a reference to the parent object is passed in. The ShowAnnotationForm routine displays the Annotation form and enables you to add, edit, and delete the annotation from a parent object record. For example, to display the Annotation form on an account, you would create and load a CGLAccount object and pass it to ShowAnnotationForm. For more information about service objects, see "Understanding Service Objects" on page 14. For more information about working with child objects, see "Working with Objects" on page 14.

The following code sample illustrates adding an Annotation form to an account record.

```
Private Sub ShowAnnotationForm(oAccount as CGLAccount)

    Dim oAnnotationFrm As CAnnotationForm
    Set oAnnotationFrm = goFE_Services.CreateServiceObject(bbsoAnnotationForm)

    'Load the annotation form with the selected data object
    With oAnnotationFrm
        .Init goFE_API.SessionContext
        .ShowAnnotationForm oAccount, Me
        .CloseDown
    End With
    Set oAnnotationFrm = Nothing

    'Save the data object with the new annotation
    With oAccount
        .Save
        .CloseDown
    End With

End Sub
```

Sample: Managing Code Tables

Code tables in ***The Financial Edge*** provide advantages such as standardizing user input and helping to speed data entry. Two service objects give you access to tables and the ability to manipulate them in your application. The first, bbsoCodeTableServer, gives you access to code table entries for a specific code table and to static code tables. You can load table entries directly into combo boxes or you can retrieve a variant array containing the table entry descriptions and their numeric IDs. The second object, bbsoTableLookupServer, provides access to code tables through the standard code table lookup form. From this form, users can add, edit, delete, and select code table entries.

Using bbsoCodeTableServer

With bbsoCodeTableServer, you can return ***Financial Edge*** code table entries, their IDs, and descriptions. In the following code sample, LoadCodeTableCombo loads a combo called cboCodeTable with the Net Asset Class Types table entries. LoadCodeTableArray returns a two-dimensional variant array, where the first dimension has a lower bound of 0 and an upper bound of 1. The second dimension has a lower bound of 1 and an upper bound equal to the number of table entries.

```
Private Sub LoadCodeTableCombo()

    Dim oCodeTableServer As CCodeTablesServer

    Set oCodeTableServer = oFE_Services.CreateServiceObject(bbsoCodeTablesServer)

    cboCodeTable.Clear

    'This loads a combo box with Net Asset Class types
    With oCodeTableServer
        .Init oFE_API.SessionContext
        .LoadCombo cboCodeTable, ctnumGLClass
        .CloseDown
    End With

    Set oCodeTableServer = Nothing
End Sub

Private Sub LoadCodeTableArray()

    Dim oCodeTableServer As CCodeTablesServer
    Set oCodeTableServer = oFE_Services.CreateServiceObject(bbsoCodeTablesServer)

    'This loads an Array with GL Source descriptions along with their IDs
    With oCodeTableServer
        .Init oFE_API.SessionContext
```

(Continued- page 2 of 2)

```
        '.CodeTableGetDataArray: Returns a 2 dimensional variant array containing
        'the ID in vAry(0, n) and the description in vAry(1, n)

        Dim vAry As Variant
        Dim l as Long

        vAry = .CodeTableGetDataArray(ctnumGLSource)

        For l = LBound(vAry, 2) To UBound(vAry, 2)
            1stSourceArray.AddItem vAry(0,1) & "-" & vAry(1, 1)
        Next 1

        Erase vAry
        oCodeTableServer.CloseDown
    End With

    Set oCodeTableServer = Nothing

End Sub
```

Using bbsoTableLookupServer

ShowCodeTableForm displays the standard table entry form and places the return value into the label lblLookup.

```
Private Sub ShowCodeTableForm()

    Dim oTableLookupHandler As CTableLookupHandler

    Set oTableLookupHandler = oFE_Services.CreateServiceObject(bbsoTableLookupServer)

    With oTableLookupHandler
        .Init oFE_API.SessionContext
        .ShowForm ctnumGLSource, oFormToCenterOn:=Me

        If .Canceled Then
            lblLookUp.Caption = ""
        Else
            lblLookUp.Caption = "TableEntries.ID: " & .SelectedItem
        End If

        .CloseDown
    End With
    Set oTableLookupHandler = Nothing

End Sub
```

Sample: Using Grids and Controls

With FEControls.ocx, you can add common ***Financial Edge*** grids and associated controls to your application without having to recreate the grids from scratch. With FeChildGrid, you can use grids and controls to display an object's common children, such as notes, attributes, and distributions. With FeDataGrid, you can use a ***Financial Edge*** grid to display all the objects in a collection. To use FEControls.ocx, add the control to a form and initialize it with the data object and child type you want to see.

You can add FEControls7.ocx to your project. From the menu bar, select **Tools/Additional Controls**. Find FEControls7 and mark the checkbox.

The Financial Edge installation folder contains two code samples that illustrate the advantages of FEControls.OCX:

- **Example 1:** install folder\Help\Samples\Advanced samples\API\Samples\SamplesInXls\Book2.xls
- **Example 2:** install folder\Help\Samples\API\FEControls

Sample: Listing Records

To automate processes in your code, you can loop through collections of top-level objects. For example, to create a list of all accounts by their descriptions, you can loop through the CGLAccounts collection. To create any ***Financial Edge*** object, you need a valid SessionContext. The SessionContext object contains all the user's connection information and can be accessed from the FE_API property "SessionContext". For more information about the SessionContext, see "Initializing and Releasing Objects" on page 14.

The following code sample illustrates looping through the CGLAccounts collection and adding account descriptions to a list box.

```
'This example assumes the global variable goFE_API is declared and initialized
'Create a reference to a CGLAccount and CGLAccounts object
Dim oAccount as CGLAccount
Dim oAccounts as CGLAccounts

'Create an instance of the CGLAccounts object
'Using the SessionContext from the FE_API, set the bReadOnly parameter so we
'don't lock up the records
Set oAccounts = New CGLAccounts
oAccounts.Init goFE_API.SessionContext, bReadOnly:=True

'Loop through the oAccounts collection and pull out the names
For Each oAccount In oAccounts
    With lstRecords
        .AddItem oAccount.Fields(GLAccountS_fld_DESCRIPTION)
    End With
    'Only pull the first 10 records for this demonstration
    If lstRecords.ListCount > 10 Then Exit For
Next

'Clean up the object references
oAccount.CloseDown
Set oAccount = Nothing

oAccounts.CloseDown
Set oAccounts = Nothing
```

Sample: Managing Media and Notepads

Using the Media and Notepad service objects, you can add, edit, and delete media and notepad records. Both have similar properties and methods, and because they are child objects, both require a parent record that is initialized and loaded. You must also set a reference to the parent’s Media or Notepad collection. To display the Media or Notepad form, you pass the ShowForm method.

The following code sample illustrates adding new media to a project record.

```
Private Sub ShowMediaForm(oProject As CGLProject)

    'Create an instance of the CMediaForm object and create
    'the service object bbsoMediaForm
    Dim oMediaFrm As CMediaForm
    Set oMediaFrm = goFE_Services.CreateServiceObject(bbsoMediaForm)

    'Load the Media form with the selected data object
    With oMediaFrm
        .Init goFE_API.SessionContext

        'Set a reference in oMediaFrm to the collection of media
        'objects on the parent
        Set .MediaObjects = oProject.Media

        .ShowForm Me
        .CloseDown
    End With
    Set oMediaFrm = Nothing

    'Save the data object with the new media
    With oProject
        .Save
        .CloseDown
    End With

End Sub
```

Sample: Printing Reports

With the FE_Services object, you can access standard ***Financial Edge*** forms, search screens, data objects, and reports. For example, to preview all income statements created by the user “Supervisor”, you could initialize the goFE_Services object and use the CreateServiceObject method to create an instance of the IBBReportInstances object. Then, you could iterate through the IBBReportInstances collection and preview all reports generated by Supervisor. For more information about the FE_Services object, see “Managing the FE_Services Object” on page 107.

```
'Create a reference to a IBBReportInstance object
'and a IBBReportInstances collection
Dim oReport As IBBReportInstance
Dim oReports As IBBReportInstances

'Create an instance of the IBBReportInstances collection
Set oReports = FE_Services.CreateServiceObject(bbsoReportInstances)

'Initialize the collection with the SessionContext and the report type for Income Statement
oReports.Init FE_API.SessionContext, bbrep_GL_IncomeStatement

'Cycle through each report and preview them
For Each oReport In oReports
    With oReport
        .Init FE_API.SessionContext

        'Process only the reports generated by the Supervisor
        If .Property(REPORTPARAMETERNAMES7_fld_ADDEDBYNAME) = "Supervisor" Then
            .Process bbrep_ProcessOption_Preview
        End If

    End With

    oReport.CloseDown
    Set oReport = Nothing

Next

'Clean up the object references
Set oReports = Nothing
```

Sample: Using the Search Screen

A major advantage of integrating with ***The Financial Edge*** is the ability to use the features and functions in the application without having to recreate them. For example, you can display a search screen and use it to locate and open account records. To use this form, you first initialize the goFE_Services object, then use the CreateServiceObject method to create an instance of the IBBSearchScreen object. For more information about the IBBSearchScreen object, see the Object Reference section of the VBA and API help file.

```
'Create an instance of an IBBSearchScreen object and create the service object
'bbsoSearchScreen
Dim oSearch As IBBSearchScreen
Set oSearch = FE_Services.CreateServiceObject(bbsoSearchScreen)

'Create a CGLAccount object to hold the returned account
Dim oAccount as CGLAccount

With oSearch
    .Init FE_API.SessionContext

    'Look only for accounts
    .AddSearchType SEARCH_GLACCOUNT

    .ShowSearchForm

    If .SelectedID > 0 Then
        Set oAccount = .SelectedDataObject
        lstRecords.AddItem oAccount.Fields(GLACCOUNTS_fld_DESCRIPTION)
        oAccount.CloseDown
        Set oAccount = Nothing
    End If

End With

'Clean up the object references
oSearch.CloseDown
Set oSearch = Nothing
```

Sample Programs

Chapter 5

Contents

This chapter contains code samples for creating applications with *API*, *VBA*, and *Read-Only Database Assistance*. For additional sample programs, see the The_Financial_Edge\Help\Samples folder.

API Samples

Several API sample programs come installed in The_Financial_Edge\Help\Samples\API folder. These samples consist of two levels of expertise, beginner and intermediate.

The following code samples provide the basics of connecting to ***The Financial Edge***.

Sample	Format	Description
Log In	*Visual Basic 6.0*	Demonstrates three methods for connecting to a database: connecting directly with no user interaction, prompting the user for login name and password using the standard ***Financial Edge*** login screen, and using a custom login screen.
Search Screen	*Visual Basic 6.0*	Uses the standard ***Financial Edge*** search screen to lookup account and project records.
List Records	*Visual Basic 6.0*	Using the CGLAccounts object, retrieves a list of account descriptions.

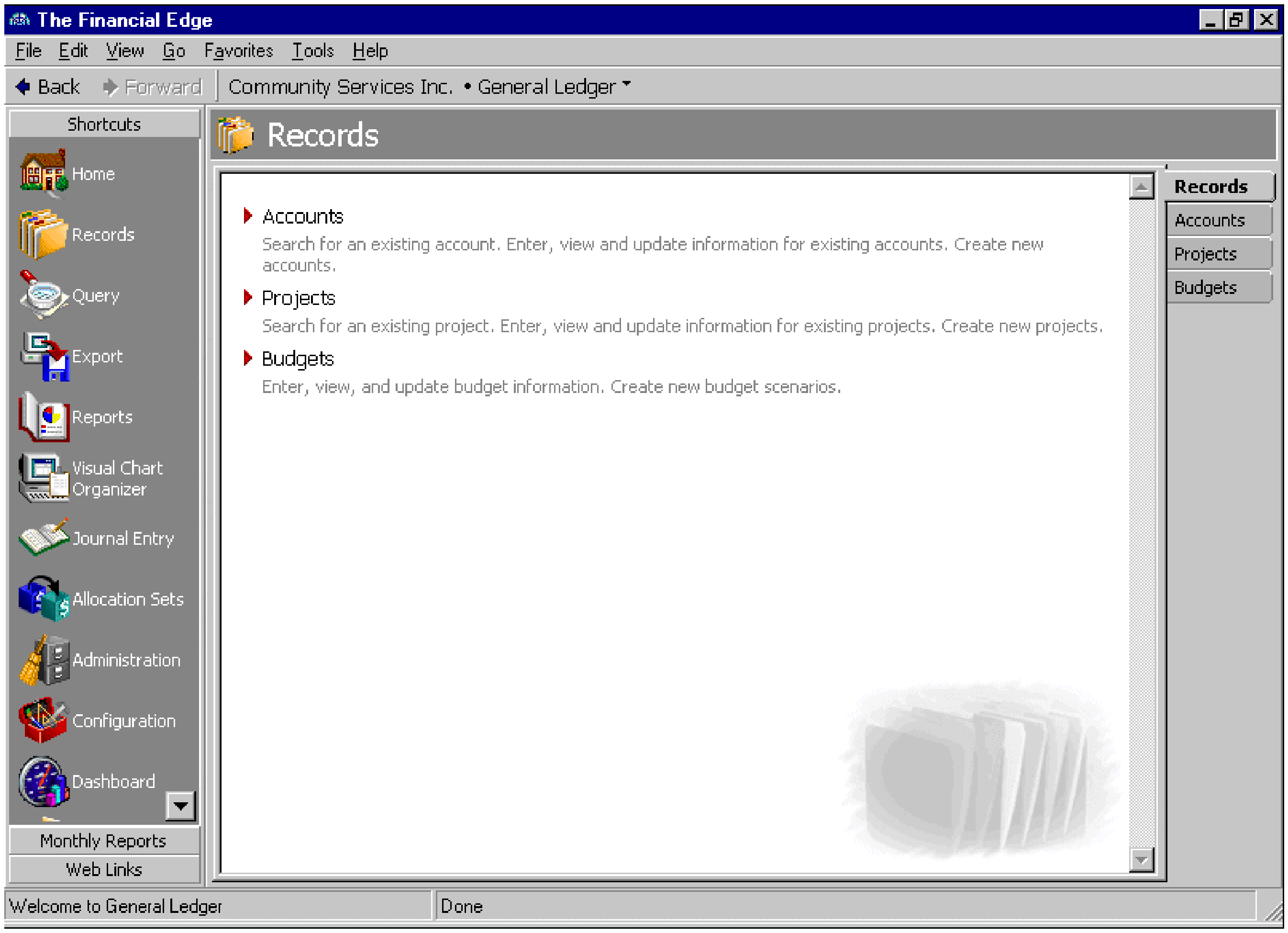
The following code samples, using advanced API features, are slightly more complex.

Sample	Format	Description
Attribute Types	*Visual Basic 6.0*	Demonstrates the AttributeTypeServer object.
Code Table Server	*Visual Basic 6.0*	Shows how to extract code table information. It also demonstrates the TableLookupHandler object, which displays the standard code table form.
Forms	*Visual Basic 6.0*	Uses the CreateServiceObject method of the FE_Services object to display a Search Screen, Annotation, Media and Notepad forms.
Misc UI	*Visual Basic 6.0*	Displays several of the miscellaneous user interface forms, including Print Setup and the About form. Makes use of the *QuickFind* method of MiscUI interface to find records (an alternative to the standard search screen).
Prog IDs	*Visual Basic 6.0*	Demonstrates an alternate method for creating a data object using the *GetProgIDforDataObject* method of the FE_Services object.
Reports	*Visual Basic 6.0*	Shows how to automate running reports through the API.
Treeview of Reports	*Visual Basic 6.0*	Provides an example of how to display reports in a treeview control, which is a component available with the *Microsoft Windows Common Controls 6.0* (mscomctl.ocx). To use the treeview control, select **Project**, **Components**.

Additionally, this guide contains API code samples for records and reports in ***General Ledger***, ***Accounts Payable***, ***Fixed Assets***, ***Accounts Receivable***, ***Cash Receipts,*** and ***Student Billing*** and common samples used throughout ***The Financial Edge***.

General Ledger Records Samples

This section contains code samples for creating applications you can use with ***General Ledger*** records. Samples include adding accounts, projects, and budgets, and configuring accounts.



Accounts
Search for an existing account. Enter, view and update information for existing accounts. Create new accounts.

Projects
Search for an existing project. Enter, view and update information for existing projects. Create new projects.

Budgets
Enter, view, and update budget information. Create new budget scenarios.

Creating a General Ledger Account Structure

The following code sample provides a method for configuring ***General Ledger*** accounts from a third-party application.

```
'Initial configuration of the Account involves creating the following
'Account_Structure
'Category_Definitions
'Defining Account Codes

'Sets the properties of the Fund and AccountCode Segments of the AccountStructure

Public Function Create_Required_Segments(ByVal sFundlength As String, ByVal sSeparator _
                                         As String, ByVal sAccountlength As String)

    sSeparator = Get_Separator (sSeparator)

    Dim sFundType As String
    Dim sAccountType As String

    With goCodeTablesServer
        sFundType = .StaticTableTranslation(staticnumGLSegmentType, _
          staticentryGLSegmentType_Fund)
        sAccountType = .StaticTableTranslation(staticnumGLSegmentType, _
          staticentryGLSegmentType_AccountCode)
    End With

    Dim oSegments As cGLSegments
    Set oSegments = New cGLSegments

    With oSegments
        .Init goSessionContext
        With .Add
            .Fields(GLSEGMENTS_fld_TYPE) = sFundType
            .Fields(GLSEGMENTS_fld_LENGTH) = sFundlength
            .Fields(GLSEGMENTS_fld_SEPARATOR) = sSeparator
            .Fields(GLSEGMENTS_fld_SEQUENCE) = 1 'Fund is the first segment
        End With

        With .Add
            .Fields(GLSEGMENTS_fld_TYPE) = sAccountType
            .Fields(GLSEGMENTS_fld_LENGTH) = sAccountlength
            .Fields(GLSEGMENTS_fld_SEPARATOR) = "" 'Separator for account is blank
            .Fields(GLSEGMENTS_fld_SEQUENCE) = 2 'Account is the second segment
        End With
        .Save
        .CloseDown
    End With
    Set oSegments = Nothing

End Function
'makes an entry in the Segments Table for the segment to be created
'and links it to the codeTable
```

(Continued- page 2 of 6)

```
Friend Function Create_Segment(ByVal sSegName As String, ByVal sSegSep As String, _
                               ByVal sSegLength As String)

    Dim sSegType As String
    sSegType = goCodeTablesServer.StaticTableTranslation(staticnumGLSegmentType, _
      staticentryGLSegmentType_Table)

    sSegSep = Get_Separator(sSegSep)

    'GetTableId function returns TableId if entry exists with the same name as the segment
    'It returns 0 if the entry does not exist
    If GetTableId(sSegName) = 0 Then

    'Creates an entry in CodeTables for the new segment to be added
        Create_Table sSegName, sSegLength

        Dim lNumOfSeg As Long
        Dim oSegments As cGLSegments
        Set oSegments = New cGLSegments
        With oSegments
            .Init goSessionContext
            lNumOfSeg = .Count 'Gets the Current Number of Segments
            .Item(lNumOfSeg).Fields(GLSEGMENTS_fld_SEPARATOR) = sSegSep
            With .Add
                .Fields(GLSEGMENTS_fld_TYPE) = sSegType
                .Fields(GLSEGMENTS_fld_CODETABLESID) = GetTableId(sSegName)
                .Fields(GLSEGMENTS_fld_LENGTH) = sSegLength
                .Fields(GLSEGMENTS_fld_SEPARATOR) = ""
                .Fields(GLSEGMENTS_fld_SEQUENCE) = lNumOfSeg + 1
            End With
            .Save
            .CloseDown
        End With
        Set oSegments = Nothing
    End If
End Function

'This function checks if a segment of the name passed to it already exists
'Returns the 'TableId' if segment exists or 0 if segment doesn't exist

Private Function GetTableId(ByVal sSegName As String) As Long
    Dim oCodeTables As CCodeTables
    Dim oCodeTable As CCodeTable

    Set oCodeTables = New CCodeTables
    oCodeTables.Init goSessionContext

    Dim lTableId As Long
    For Each oCodeTable In oCodeTables
        If oCodeTable.Fields(ctfNAME) = sSegName Then
            lTableId = oCodeTable.Fields(ctfCODETABLEID)
```

(Continued- page 3 of 6)

```
            Exit For
        End If
    Next oCodeTable
    oCodeTables.CloseDown
    Set oCodeTables = Nothing
    GetTableId = lTableId
End Function

'Create a new code table with the same name as the segment
Private Sub Create_Table(ByVal sSegName As String, ByVal sSegLength As String)
    Dim oCodeTable As CCodeTable
    Set oCodeTable = New CCodeTable

    With oCodeTable
        .Init goSessionContext
        .Fields(ctfACTIVE) = True
        .Fields(ctfSHORTDESCLENGTH) = Val(sSegLength)
        .Fields(ctfHASSHORTDESC) = True
        .Fields(ctfNAME) = sSegName
        .Fields(ctfUSERDEFINED) = True
        .Fields(ctfSYSTEMMASK) = bbBlackbaud_GL_System
        .Save
        .CloseDown
    End With
    Set oCodeTable = Nothing
End Sub

'Define Category Ranges
Friend Function Define_Category_Ranges( _
     bAsset As Boolean, lAssetRangeStart As Long, lAssetRangeEnd As Long, _
     bLiability As Boolean, lLiabilityRangeStart As Long, lLiabilityRangeEnd As Long, _
     bNetAsset As Boolean, lNetAssetRangeStart As Long, lNetAssetRangeEnd As Long, _
     bRevenue As Boolean, lRevenueRangeStart As Long, lRevenueRangeEnd As Long, _
     bExpense As Boolean, lExpenseRangeStart As Long, lExpenseRangeEnd As Long, _
     bGift As Boolean, lGiftRangeStart As Long, lGiftRangeEnd As Long, _
     bTransfer As Boolean, lTransferRangeStart As Long, lTransferRangeEnd As Long, _
     bGain As Boolean, lGainRangeStart As Long, lGainRangeEnd As Long, _
     bLoss As Boolean, lLossRangeStart As Long, lLossRangeEnd As Long)

'If the category is not set to true (meaning it is enables or checked) then it doesn't
'matter what long value you enter for its category ranges.
    Dim sAsset As String
    Dim sLiability As String
    Dim sNetAsset As String
    Dim sRevenue As String
    Dim sExpense As String
    Dim sGift As String
    Dim sTransfer As String
    Dim sGain As String
    Dim sLoss As String
```

(Continued- page 4 of 6)

```
    With goCodeTablesServer
        sAsset = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Asset)
        sLiability = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Liability)
        sNetAsset = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_NetAsset)
        sRevenue = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Revenue)
        sExpense = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Expense)
        sGift = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Gift)
        sTransfer = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Transfer)
        sGain = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Gain)
        sLoss = .StaticTableTranslation(staticnumGLCategoryType, _
          staticentryGLCategoryType_Loss)
    End With

    Dim oCategories As cGLCategoryDefinitions
    Set oCategories = New cGLCategoryDefinitions
    oCategories.Init goSessionContext

    Dim oCategory As cGLCategoryDefinition

    For Each oCategory In oCategories
        With oCategory
            Select Case .Fields(GLCATEGORYDEFINITIONS_fld_CATEGORY)
                Case sAsset
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bAsset
                    If bAsset Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lAssetRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lAssetRangeEnd
                    End If
                Case sLiability
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bLiability
                    If bLiability Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lLiabilityRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lLiabilityRangeEnd
                    End If
                Case sNetAsset
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bNetAsset
                    If bNetAsset Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lNetAssetRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lNetAssetRangeEnd
                    End If
                Case sRevenue
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bRevenue
                    If bRevenue Then
```

(Continued- page 5 of 6)

```
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lRevenueRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lRevenueRangeEnd
                    End If
                Case sExpense
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bExpense
                    If bExpense Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lExpenseRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lExpenseRangeEnd
                    End If
                Case sGift
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bGift
                    If bGift Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lGiftRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lGiftRangeEnd
                    End If
                Case sTransfer
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bTransfer
                    If bTransfer Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lTransferRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lTransferRangeEnd
                    End If
                Case sGain
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bGain
                    If bGain Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lGainRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lGainRangeEnd
                    End If
                Case sLoss
                    .Fields(GLCATEGORYDEFINITIONS_fld_ENABLED) = bLoss
                    If bLoss Then
                        .Fields(GLCATEGORYDEFINITIONS_fld_FROMCODE) = lLossRangeStart
                        .Fields(GLCATEGORYDEFINITIONS_fld_TOCODE) = lLossRangeEnd
                    End If
            End Select
        End With
    Next oCategory

    With oCategories
        .Save
        .CloseDown
    End With

    Set oCategories = Nothing
    Set oCategory = Nothing

End Function

'Define Account Codes
Friend Function Define_Account_Codes( _
                            sAccountCode As String, sAccountDescription As String, _
                            sAccountCashflow As String, sAccountWorkingCapital As String, _
```

(Continued- page 6 of 6)

```
                            sAccountClass As String, bAccountControl As Boolean, _
                            bAccountContra As Boolean)
    Dim oAccountCode As CGLAccountCode
    Set oAccountCode = New CGLAccountCode

    With oAccountCode
        .Init goSessionContext
        .Fields(GLACCOUNTCODES_fld_ACCOUNTCODE) = sAccountCode
        .Fields(GLACCOUNTCODES_fld_DESCRIPTION) = sAccountDescription
        .Fields(GLACCOUNTCODES_fld_CASHFLOW) = sAccountCashflow
        .Fields(GLACCOUNTCODES_fld_WORKINGCAPITAL) = sAccountWorkingCapital
        .Fields(GLACCOUNTCODES_fld_CLASS) = sAccountClass
        .Fields(GLACCOUNTCODES_fld_CONTROLACCOUNT) = bAccountControl
        .Fields(GLACCOUNTCODES_fld_CONTRAACCOUNT) = bAccountContra
        .Save
        .CloseDown
    End With
    Set oAccountCode = Nothing
End Function

'Returns the Separator in the format required by FE
Friend Function Get_Separator(ByRef sSeparator As String) As String
    With goCodeTablesServer
        Select Case sSeparator
            Case "Hyphen"
                sSeparator = .StaticTableTranslation( _
                  staticnumGLSegmentSeparator, staticentryGLSegmentSeparator_Hyphen)
            Case "Comma"
                sSeparator = .StaticTableTranslation( _
                  staticnumGLSegmentSeparator, staticentryGLSegmentSeparator_Comma)
            Case "Semicolon"
                sSeparator = .StaticTableTranslation( _
                  staticnumGLSegmentSeparator, staticentryGLSegmentSeparator_Semicolon)
            Case "Slash"
                sSeparator = .StaticTableTranslation( _
                  staticnumGLSegmentSeparator, staticentryGLSegmentSeparator_Slash)
            Case "Backslash"
                sSeparator = .StaticTableTranslation(staticnumGLSegmentSeparator,
                  staticentryGLSegmentSeparator_Backslash)
            Case "Period"
                sSeparator = .StaticTableTranslation( _
                  staticnumGLSegmentSeparator, staticentryGLSegmentSeparator_Period)
            Case Else
                sSeparator = .StaticTableTranslation( _
                  staticnumGLSegmentSeparator, staticentryGLSegmentSparator_None)
        End Select
    End With
    Get_Separator = sSeparator

End Function
```

Adding an Account with a Default Note

With this code sample, you can create a new ***General Ledger*** account and add a default notepad to the account record.

```
'This code sample creates a new account
'This code sample works only with the Configuration defined in the
'Sample Database provided with FE

Friend Function createNewAccount(ByVal sAccNum As String, sAccDesc As String, _
                                 ByVal sAccStatus As String) As Long

'sAccNum = "01-8000-01"
'sAccDesc = "Description of The Account"
'sAccStatus = "Active"

    Dim oAccount As CGlAccount
    Set oAccount = New CGlAccount

    With oAccount
        .Init goSessionContext
        'Sets the Account-Number property of the New Account
        'Account-Number is a Required and Unique Field
        .Fields(GLACCOUNTS_fld_ACCOUNTNUMBER) = sAccNum

        'Sets the Description property of the New Account
        'Description is a Required Field
        .Fields(GLACCOUNTS_fld_DESCRIPTION) = sAccDesc

        'Sets the Status property of the New Account
        'Status is a Required Field with default as 'Active'
        .Fields(GLACCOUNTS_fld_STATUS) = sAccStatus

        'Save the account details into the database
        'Validations will take place on this statement
        .Save

        'Return the Id of the Account
        createNewAccount = .Fields(GLACCOUNTS_fld_GL7ACCOUNTSID)

        'Close the Object. So that it can be used by some other function
        .CloseDown
    End With
    Set oAccount = Nothing
End Function

'Creates a note and adds it to the Account Object whose Id is passed to it.
Friend Sub createNote(ByVal lAccountId As Long, ByVal sNoteAuthor As String, _
                            ByVal sNoteDesc As String, ByVal dtNoteDate As Date, _
                            ByVal sNoteType As String, ByVal sNoteTitle As String, _
                            ByVal sActualNotes As String, ByVal sNotes As String)
```

(Continued- page 2 of 3)

```
'sNoteAuthor = "John Wright"
'sNoteDesc = "Description of the Note"
'dtNoteDate = Date
'sNoteType = "Internal"
'sNoteTitle = "Title of the note"
'sActualNotes = "This is just a sample note"
'sNotes = "Detailed Notes About The Account"

    Dim oAcct As CGlAccount
    Set oAcct = New CGlAccount

    With oAcct
        .Init goSessionContext
        .Load lAccountId
    'set the various properties of the note
        With .Notepads.Add
            .Fields(NOTEPAD_fld_Author) = sNoteAuthor
            .Fields(NOTEPAD_fld_Description) = sNoteDesc
            .Fields(NOTEPAD_fld_NotepadDate) = dtNoteDate
            .Fields(NOTEPAD_fld_NotepadType) = sNoteType
            .Fields(NOTEPAD_fld_Title) = sNoteTitle
            .Fields(NOTEPAD_fld_ActualNotes) = sActualNotes
            .Fields(NOTEPAD_fld_Notes) = sNotes
        End With
    'Save the note details by saving the CGLAccount object
    'Validations will take place on this statement
        .Save
        .CloseDown
    End With
    Set oAcct = Nothing
End Sub


'This function returns the Account Id of the Account if it exists, 0 if it doesn't
'It takes the Account Id as the parameter

Friend Function GetAccountIdFromDescription(ByVal sAccountName As String) As Long
    Dim oAccounts As cGLAccounts
    Set oAccounts = New cGLAccounts
    oAccounts.Init goSessionContext, "DESCRIPTION LIKE '" & sAccountName & "'", True

    Dim lAccountId As Long

    Dim oAccount As CGlAccount

    For Each oAccount In oAccounts
        If oAccount.Fields(GLACCOUNTS_fld_DESCRIPTION) = sAccountName Then
            lAccountId = oAccount.Fields(GLACCOUNTS_fld_GL7ACCOUNTSID)
            oAccount.CloseDown
            Exit For
        End If
```

(Continued- page 3 of 3)

```
    oAccount.CloseDown
    Next oAccount
    'clean up
    Set oAccount = Nothing
    oAccounts.CloseDown
    Set oAccounts = Nothing
    GetAccountIdFromDescription = lAccountId
End Function


'Displays the Account Form for the account whose Id is passed to it
Friend Function displayAccountForm(ByVal lAccountId As Long) As Long
    Dim oAccount As CGlAccount
    Set oAccount = New CGlAccount
    With oAccount
        .Init goSessionContext
        .Load lAccountId
    End With

    Dim oAccountForm As cGLAccountForm
    Set oAccountForm = New cGLAccountForm
    With oAccountForm
        .Init goSessionContext
        Set .AccountObject = oAccount
        .ShowForm True, , False
        .CloseDown
    End With

    Set oAccount = Nothing
    Set oAccountForm = Nothing
End Function
```

Adding a Project

With this code sample, you can add a project, complete with contacts, notes, media, actions, and attributes.

```
'Includes functions to
'1. Create A New Project
'2. Add Contact Info to a Project
'3. Add Project Notes
'4. Add Project Media
'5. Add Project Action
'6. Add Project Attributes

'This function creates a new project
Friend Function Create_New_Project( _
                       ByVal sProjectId As String, ByVal sProjectDescription As String, _
                       ByVal sProjectStartDate As String, ByVal sProjectEndDate As String, _
                       ByVal sProjectType As String, ByVal sProjectStatus As String) As Long

    Dim oProject As CGLProject
    Set oProject = New CGLProject

    ' sProjectID = "1090"
    ' sProjectDescription = "Smith Grant"
    ' sProjectStartDate = "07/10/2001"
    ' sProjectEndDate = "07/09/2007"
    ' sProjectType = "Grant"
    ' sProjectStatus = "In Progress"

    With oProject
        .Init goSessionContext
        .Fields(GLPROJECTS_fld_PROJECTID) = sProjectId
        .Fields(GLPROJECTS_fld_DESCRIPTION) = sProjectDescription
        .Fields(GLPROJECTS_fld_STARTDATE) = sProjectStartDate
        .Fields(GLPROJECTS_fld_ENDDATE) = sProjectEndDate
        .Fields(GLPROJECTS_fld_TYPE) = sProjectType
        .Fields(GLPROJECTS_fld_STATUS) = sProjectStatus
        .Save
        lProjectId = .Fields(GLPROJECTS_fld_GL7PROJECTSID)
        .CloseDown
    End With
    Set oProject = Nothing
    Create_New_Project = lProjectId
End Function

'The function adds a new contact to the project passed to it
Friend Function Add_Project_Contact( _
            ByVal lProjectId, ByVal sProjectContactName As String, _
            ByVal sProjectOrganization As String, ByVal sProjectContactPosition As String, _
            ByVal sPhoneType As String, ByVal sPhoneNumber As String)

    'sProjectContactName = "Mr. Perry Brown"
    'sProjectContactPosition = "Regional Manager"
    'sProjectOrganization = "Wachovia"
```

(Continued- page 2 of 5)

```
    ' sProjectPhoneType = "Home"
    ' sProjectPhoneNumber = "9198888861"

With oProject
        .Init goSessionContext
        .Load lProjectId

        With .Contacts.Add
            .Fields(GLPROJECTCONTACTS_fld_ORGANIZATION) = sProjectOrganization
            .Fields(GLPROJECTCONTACTS_fld_CONTACTPOSITION) = sProjectContactPosition
            With .NameObject
                .Fields(NAME_fld_FULLNAME) = sProjectContactName
            End With

            With .Phones.Add
                .Fields(Phone_fld_PhoneType) = sPhoneType
                .Fields(Phone_fld_Num) = sPhoneNumber
            End With
        End With
        .Save
        .CloseDown
    End With
End Function


Friend Sub Add_Project_Note(ByVal lProjectId As Long, ByVal sType As String, _
                            ByVal sTitle As String, ByVal sDescription As String, _
                            ByVal sNotes As String)

    'sNoteType = "Internal"
    'sNoteTitle = "Important Note"
    'sNoteDesc = "Weekly Note By Manager"
    'sNotes = "The project is running on schedule. More manpower needed in future."

    Dim oProject As CGLProject
    Set oProject = New CGLProject

    With oProject
        .Init goSessionContext
        .Load lProjectId
        With .Notepads.Add
            .Fields(NOTEPAD_fld_NotepadType) = sType
            .Fields(NOTEPAD_fld_Title) = sTitle
            .Fields(NOTEPAD_fld_Description) = sDescription
            .Fields(NOTEPAD_fld_Notes) = sNotes
        End With
        .Save
        .CloseDown
    End With
    Set oProject = Nothing
End Sub
```

(Continued- page 3 of 5)

```
'This function creates a new note and adds it to the project

Friend Sub Add_Project_Media(ByVal lProjectId As Long, ByVal sType As String, _
                    ByVal sAuthor As String, ByVal sTitle As String, ByVal sDesc As String)

    'sMediaType = "Guidelines"
    'sAuthor = "Supervisor"
    'sMediaTitle = "Insert Picture"
    'sMediaDesc = "Some pictures for this project."

    Dim oProject As CGLProject
    Set oProject = New CGLProject

    With oProject
        .Init goSessionContext
        .Load lProjectId
        With .Media.Add
            .Fields(MEDIA_fld_MediaType) = sType
            .Fields(MEDIA_fld_Author) = sAuthor
            .Fields(MEDIA_fld_Title) = sTitle
            .Fields(MEDIA_fld_Description) = sDesc
            '.Fields(MEDIA_fld_Object) = "" 'link to a media object
        End With
        .Save
        .CloseDown
    End With
    Set oProject = Nothing
End Sub

'This function creates a new Action and adds it to the project
Friend Sub Add_Project_Action(ByVal lProjectId As Long, ByVal sActionType As String, _
    ByVal dtActionDate As Date, ByVal sActionTime As String, _
    ByVal sActionStatus As String, ByVal sPriority As String, _
    ByVal sAssignedToId As String, ByVal sDescription As String, _
    ByVal bAutoRemind As Boolean, ByVal sRemindUserId As String, _
    ByVal sRemindNumUnits As String, ByVal sRemindUnitType As String, _
    ByVal bActionCompleted As Boolean, ByVal dtDateCompleted As Date)

    'sActionType = "Meeting"
    'dtActionDate = Date
    'sActionTime = Hour(Time) & ":" & Minute(Time) & ":" & Second(Time)
    'sActionStatus = "Pending"
    'sPriority = "High"
    'sAssignedToId = "Supervisor"
    'sDescription = "Adding an action to the current project"
    'bAutoRemind = True
    'sRemindUserId = "Supervisor"
    'sRemindNumUnits = "1"
    'sRemindUnitType = "Days"
    'bActionCompleted = True
    'dtDateCompleted = Date
```

(Continued- page 4 of 5)

```
    Dim oProject As CGLProject
    Set oProject = New CGLProject

    With oProject
        .Init goSessionContext
        .Load lProjectId
        With .Actions.Add
            'If the sActionType does not exist in the Table of ActionType, create an entry
            'CreateTableEntry ctnumGLActionType,sActionType,True
            .Fields(ACTIONS_fld_ACTIONTYPE) = sActionType
            .Fields(ACTIONS_fld_ACTIONDATE) = dtActionDate
            .Fields(ACTIONS_fld_ACTIONTIME) = sActionTime
            'If TableEntry does not exist then
            'CreateTableEntry ctnumGLActionStatus, sActionStatus, True
            .Fields(ACTIONS_fld_ACTIONSTATUS) = sActionStatus
            .Fields(ACTIONS_fld_PRIORITY) = sPriority
            .Fields(ACTIONS_fld_ASSIGNEDTOID) = sAssignedToId
            .Fields(ACTIONS_fld_DESCRIPTION) = sDescription
            .Fields(ACTIONS_fld_AUTOREMIND) = bAutoRemind
            .Fields(ACTIONS_fld_REMINDUSERID) = sRemindUserId
            .Fields(ACTIONS_fld_REMINDNUMUNITS) = sRemindNumUnits
            .Fields(ACTIONS_fld_REMINDUNITTYPE) = sRemindUnitType
            .Fields(ACTIONS_fld_ACTIONCOMPLETED) = bActionCompleted
            .Fields(ACTIONS_fld_DATECOMPLETED) = dtDateCompleted
            .Save
        End With
        .Save
        .CloseDown
    End With
    Set oProject = Nothing
End Sub


'This function creates a new attribute and adds it to the project
Friend Sub Add_Project_Attribute(ByVal lProjectId As Long, ByVal sAttribId As String, _
                                 ByVal dtAttribDate As Date, ByVal sComments As String, _
                                 ByVal sAttribDesc As String, ByVal sShortAttribDesc)

    'sAttribId = "Grant Amount"
    'Grant amount attribute must exist in the Project Attributes table for this code to work
    'dtAttribDate = Date
    'sShortAttribDesc = "Total Grant"
    'sAttribDesc = "100000"
    'sComments = "Grant Manager - Bob Smith"
    'sValue = "Bob Smith"
    Dim oProject As CGLProject
    Set oProject = New CGLProject

    With oProject
        .Init goSessionContext
        .Load lProjectId
```

(Continued- page 5 of 5)

```
        With .Attributes.Add
            .Fields(Attribute_fld_ATTRIBUTETYPESID) = getAttributeTypeId(sAttribId, _
                                                    bbGlobalAttributeType_GLProject)
            .Fields(Attribute_fld_ATTRIBUTEDATE) = dtAttribDate
            .Fields(Attribute_fld_COMMENTS) = sComments
            .Fields(Attribute_fld_VALUE) = sAttribDesc
        End With
        .Save
        .CloseDown
    End With
    Set oProject = Nothing
End Sub


Public Function CreateTableEntry(ByVal lTableNum As Long, ByVal sEntryDesc As String, ByVal
bActive As Boolean)
    Dim oTableEntry As CTableEntry
    Set oTableEntry = New CTableEntry

    With oTableEntry
        .Init goSessionContext, True
        .TableNumber = lTableNum
        .Fields(tableentry_fld_ACTIVE) = bActive
        .Fields(tableentry_fld_DESCRIPTION) = sEntryDesc
        .Save
        .CloseDown
    End With
    Set oTableEntry = Nothing
End Function
```

Adding a Budget

With this code sample, you can add a budget to ***General Ledger***.

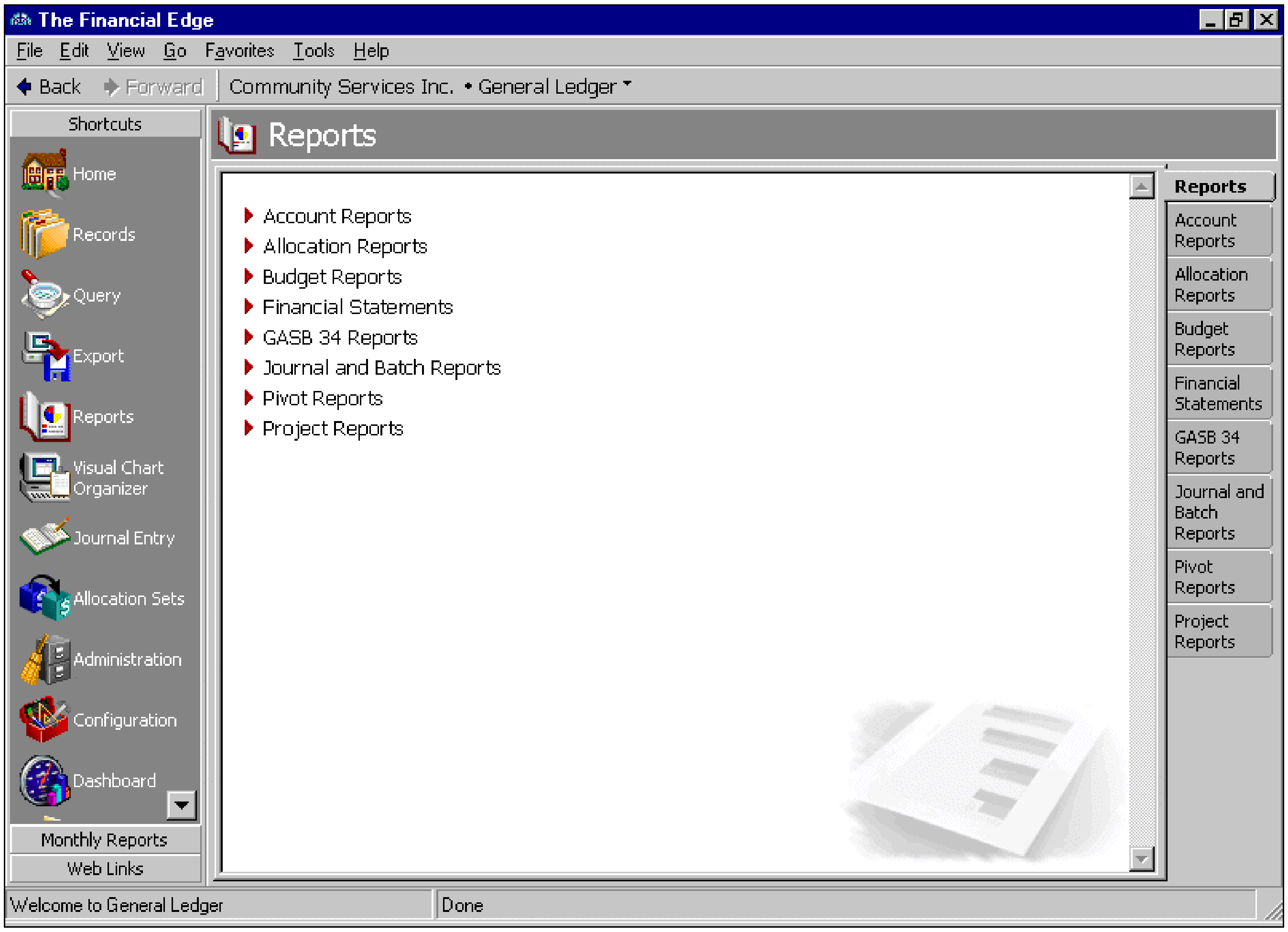
```
Friend Sub AddBudget(ByVal dtBudgetDate As Date, ByVal sScenarioId As String, _
    ByVal sScenarioDescription As String, ByVal sBudgetType As String, _
    ByVal sAccountBudgetId As String, ByVal dAccountBudgetAmount As Double, _
    ByVal dBudgetDetailAmount As Double, ByVal sProjectBudgetId As String, _
    ByVal sTCodeID As String, ByVal dProjectBudgetAmount As Double, _
    ByVal dProjectBudgetDetailAmount As Double)

    'dtBudgetDate = "04-01-2000"
    'sScenarioId = "12"
    'sScenarioDescription = "10% Above Projected Cost"
    'sBudgetType = "Fiscal Year"
    'sAccountBudgetId = "01-1030-00"
    'dAccountBudgetAmount = 1200
    'dBudgetDetailAmount = 100
    'sProjectBudgetId = "1001"
    'sTCodeID = "Elder Care"
    'dProjectBudgetAmount = 120
    'dProjectBudgetDetailAmount = 10

    Dim oBudgetScenario As CGLBudgetScenario
    Set oBudgetScenario = New CGLBudgetScenario
    With oBudgetScenario
        .Init goSessionContext

        'Set the fiscal year and scenario fields
        .Fields(GLBUDGETSCENARIOS_fld_GL7FISCALYEARSID) = _
        goFE_Service.GLFiscalYearFromDate(dtBudgetDate)
        .Fields(GLBUDGETSCENARIOS_fld_GL7BUDGETSCENARIOSID) = _
        AddScenarioTableEntry(sScenarioDescription, sScenarioId)
        .Fields(GLBUDGETSCENARIOS_fld_SCENARIODESCRIPTION) = sScenarioDescription
        .Fields(GLBUDGETSCENARIOS_fld_BUDGETTYPE) = sBudgetType
        With .AccountBudgets.Add()
        'Add required fields
            .Fields(GLACCOUNTBUDGETS_fld_GL7ACCOUNTSID) = sAccountBudgetId
            .Fields(GLACCOUNTBUDGETS_fld_AMOUNT) = dAccountBudgetAmount

            'Set Account Budget Detail amounts
            Dim oBudgetDetail As CGLAccountBudgetDetail
            For Each oBudgetDetail In .BudgetDetails
                oBudgetDetail.Fields(GLACCOUNTBUDGETDETAILS_fld_AMOUNT) = _
                dBudgetDetailAmount
            Next oBudgetDetail

            'Create a project budget
            With .ProjectBudgets.Add()
                .Fields(GLPROJECTBUDGETS_fld_GL7PROJECTSID) = sProjectBudgetId
                .Fields(GLPROJECTBUDGETS_fld_TCODEID) = sTCodeID
                .Fields(GLPROJECTBUDGETS_fld_AMOUNT) = dProjectBudgetAmount
```

(Continued- page 2 of 2)

```
             'set project budget detail amounts
                 Dim oProjectDetail As CGLProjectBudgetDetail
                 For Each oProjectDetail In .BudgetDetails
                     oProjectDetail.Fields(GLPROJECTBUDGETDETAILS_fld_AMOUNT) = _
                     dProjectBudgetDetailAmount
                 Next oProjectDetail
             End With
         End With
         .Save
         .CloseDown
     End With
     Set oBudgetScenario = Nothing
End Sub
```

General Ledger Reports Samples

This section contains code samples for creating applications you can use with ***General Ledger*** reports. Samples include creating trial balance, detail, balance sheet, and budget adjustment reports.



Creating a Trial Balance Report

With this code sample, you can create a ***General Ledger*** trial balance report.

```
Public Sub Create_Trial_Balance_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_GL_TrialBalanceReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Trial Balance Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the api"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With

    With oMetaData
        'Set the Various Filters

        'Set the Account Filter. Select the range of accounts to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_GL_AccountNumbers))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Two ways to set the account number. Shown Below.
            'Pass the account number directly into the 'From' field
            .Fields(FILTERS_fld_FROMVALUE) = "01-1000-00"
            'Pass the Id of the Account number into the 'To' field
            .Fields(FILTERS_fld_TOID) = goFE_Service.GLAccountIDFromNumber("01-1200-00")
        End With

        'Set the Project Filter. Select the project to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,_
                                 valuenumber:=1, ValueSet:=CStr(bbFilterType_GL_Projects))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getProjectId("Annabelle Johnson Endowment")
            '.Fields(FILTERS_fld_FROMVALUE) = "Annabelle Johnson Endowment"
        End With
```

(Continued- page 2 of 2)

```
        'Set the Classes Filter. Select the Account Class to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,_
                                 valuenumber:=1, ValueSet:=CStr(bbFilterType_GL_Classes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getIdFromCodeTable(ctnumGLClass, _
                                                             "Unrestricted Net Assets")
        End With
    End With

    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Creating a General Ledger Detail Report

This code sample creates a General Ledger detail report.

```
Public Sub Create_GL_Detail_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_GL_GeneralLedgerReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "GL Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the api"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With

    With oMetaData
        'Set the Various Filters

        'Set the Fund Filter. Select the Fund to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters,
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_GL_Funds))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Select Fund with Value '01'
            .Fields(FILTERS_fld_FROMVALUE) = "01"
        End With

        'Set the Account Code filter. Select a range of account codes you want to include
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_GL_AccountCodes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMVALUE) = "1000"
            .Fields(FILTERS_fld_TOVALUE) = "1200"
        End With
```

(Continued- page 2 of 2)

```
        'Set the Journal Filter. Select the Journal to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_GL_Journals))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Get the ID of the CodeTable 'Journal'
             .Fields(FILTERS_fld_FROMID) = searchInCodeTable("Journal")
        End With
    End With
    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating a Project Activity Report

With this code sample, you can create a project activity report.

```
Public Sub Create_Project_Activity_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_GL_ProjectActivityReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Project Activity Report From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the api"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With
    With oMetaData
        'Set the Various Filters

        'Set the Project Attributes Filter. Select the Project Attributes
        '    and Their Values to include in the report
        With oMetaData.PropertyDataObject(bbrep_Offset_Filters, _
                            bbrep_FilterParameter_FilterValues, _
                            valuenumber:=1, ValueSet:=CStr(bbFilterType_GL_ProjAttributes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_GLOBALATTRIBUTETYPE) = bbGlobalAttributeType_GLProject
            .Fields(FILTERS_fld_FROMID) = getAttribTypeId("Endowment Manager")
            .Fields(FILTERS_fld_FROMVALUE) = "Sue White"
        End With

        'Set the Account Code Filter. Select the range of account codes to include
        '    in the report
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,
_
        1, ValueSet:=CStr(bbFilterType_GL_AccountCodes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMVALUE) = "1000"
            .Fields(FILTERS_fld_TOVALUE) = "1200"
        End With
    End With
```

(Continued- page 2 of 2)

```
    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating a Balance Sheet

With this code sample, you can create a balance sheet report.

```
Public Sub Create_Balance_Sheet_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_GL_BalanceSheet)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
    .Init goSessionContext

    'Set the name of the report
    .Property(REPORTPARAMETERNAMES_fld_NAME) = "Balance Sheet Report Created From API"

    'Description of the report
    .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the api"

    'Can others execute this report ?
    .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

    'Can others modify this report ?
    .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With
    With oMetaData
        'Set the Various Filters

        'Set the Fund Filter. Select the Fund to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
            valuenumber:=1, ValueSet:=CStr(bbFilterType_GL_Funds))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Select Fund with Value '01'
            .Fields(FILTERS_fld_FROMVALUE) = "01"
        End With

        'Set the Account Code Filter. Select a range of acct codes to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_GL_AccountCodes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMVALUE) = "1000"
            .Fields(FILTERS_fld_TOVALUE) = "1200"
        End With
```

(Continued- page 2 of 3)

```
        'Set the Journal Filter. Select the Journal to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_GL_Journals))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Get the ID of the CodeTable 'Journal'
            .Fields(FILTERS_fld_FROMID) = searchInCodeTable("Journal")

        End With

        'Set the number of columns to 4
        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_Count).Fields( _
          REPORTPARAMETERVALUES_fld_NUMBER) = 4

        'Set the heading of the 4th column
        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_Heading, , _
          ValueSet:="C4").Fields(REPORTPARAMETERVALUES_fld_TEXT)= "New"

        'Set the alignment of the heading of the 4th Column
        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_HeadingAlign, , _
          ValueSet:="C4").Fields(REPORTPARAMETERVALUES_fld_NUMBER) = Cry_Alignment_Right

        'Set the Date Range of the column
        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_DateCombo, , _
          ValueSet:="C4").Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_SPECIFICDATE

        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_StartDate, , _
          ValueSet:="C4").Fields(REPORTPARAMETERVALUES_fld_DATETIME) = Now

        'Set the formula of the 4th column
        '.Fields(REPORTPARAMETERVALUES_fld_text) = "{Actual}"
        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_Formula, , _
          ValueSet:="C4").Fields(REPORTPARAMETERVALUES_fld_NUMBER) = _
          bbrep_ColumnAmountType_Actual

        'Set the number of decimals for the 4th Column
        .PropertyDataObject(bbrep_Offset_Columns, bbrep_ColumnParameter_Decimals, , _
          ValueSet:="C4").Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 3

        'Create a Multiple Column Heading
        With .PropertyDataObject(bbrep_Offset_MultiColHeadings, 1, valuenumber:=1)
            .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "Multiple Column Heading #1"
            'Set the start column for the multiple column heading
            .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1
            'Set the end column for the multiple column heading
            .Fields(REPORTPARAMETERVALUES_fld_NUMBER2) = 3
            'Set the alignment
            .Fields(REPORTPARAMETERVALUES_fld_NUMBER3) = Cry_Alignment_Left
            'Set the sequence number
            .Fields(REPORTPARAMETERVALUES_fld_SEQUENCE) = 1
```

(Continued- page 3 of 3)

```
            'As its the one and only multiple heading column
            .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = bbTrue
        'Only save values where true it turned on.
        End With
    End With

    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating a Budget Adjustment Report

With this code sample, you can create a budget adjustment report.

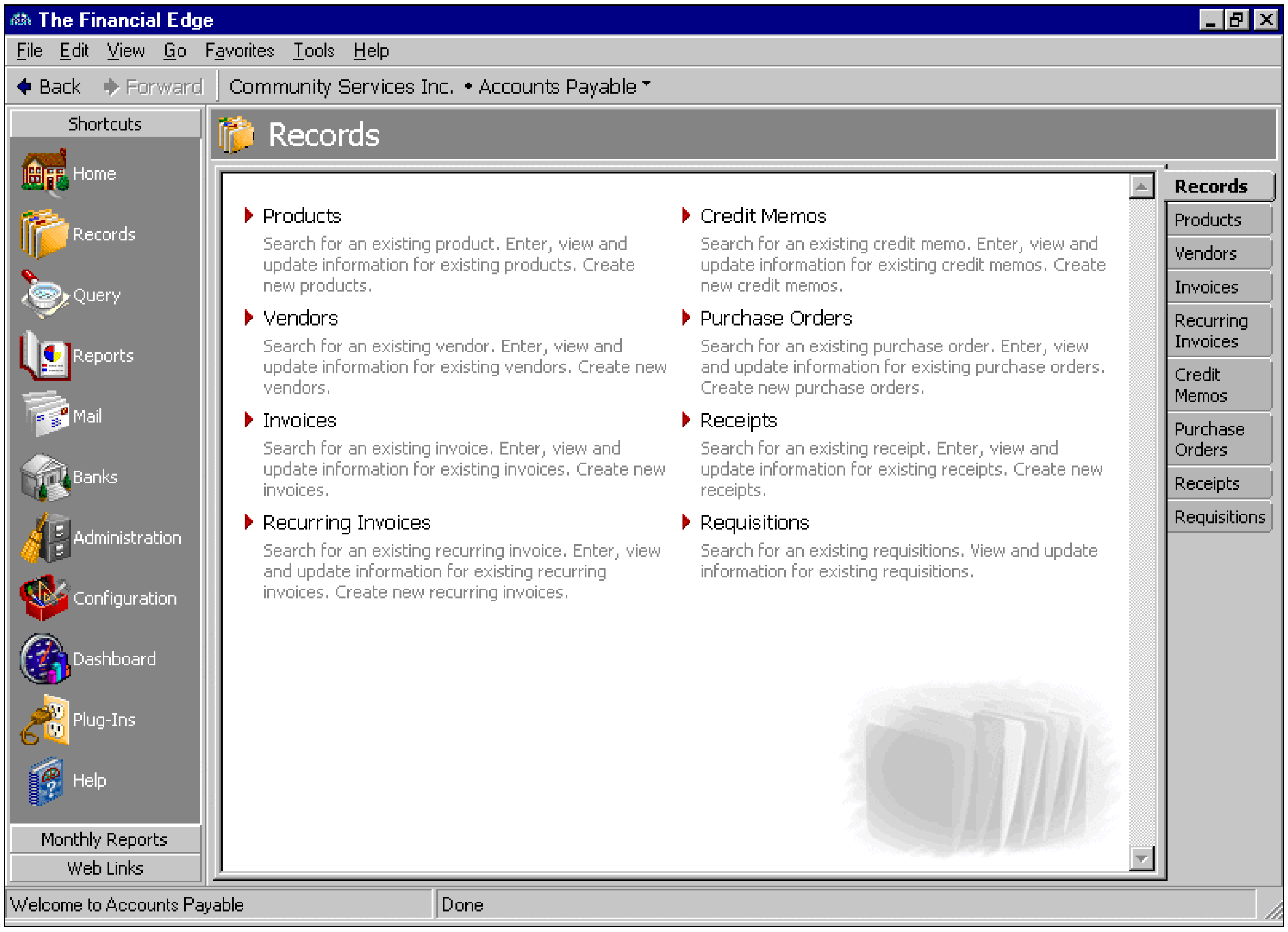
```
Public Function Create_Budget_Adjustment_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_GL_BudgetAdjustmentsReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Budget Adjustment Report Created From_
           API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the api"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData

        'Set the Various Filters

        'Set the Fund Filter. Select the Fund to include in the report
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_GL_Funds))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Select Fund with Value '01'
            .Fields(FILTERS_fld_FROMVALUE) = "01"
        End With

        'Set the Print Criteria of the format tab
        'ScenarioId, Show and ProjectAttribute fields will not be included in the report
        .PropertyDataObject(bbrep_OffSet_Format, _
          bbrep_Format_Criteria_ListOfIncludedItems,_
          bbrep_BudgetAdjustments_Criteria_ScenarioID).Fields( _
          REPORTPARAMETERVALUES_fld_BOOLEAN) = False

        .PropertyDataObject(bbrep_OffSet_Format, _
          bbrep_Format_Criteria_ListOfIncludedItems,_
          bbrep_BudgetAdjustments_Criteria_Show).Fields( _
          REPORTPARAMETERVALUES_fld_BOOLEAN) = False
```

(Continued- page 2 of 2)

```
          .PropertyDataObject(bbrep_OffSet_Format, _
            bbrep_Format_Criteria_ListOfIncludedItems,_
            bbrep_BudgetAdjustments_Criteria_ProjAttributes).Fields( _
            REPORTPARAMETERVALUES_fld_BOOLEAN) = False
     End With

     'Show the Parameter Form and Close the Report
     With oReport
          .Process bbrep_ProcessOption_ShowParameterForm, True
          .CloseDown
     End With
     Set oReport = Nothing
     Set oMetaData = Nothing
End Function
```

Accounts Payable Records Samples

This section contains code samples for creating applications you can use with ***Accounts Payable*** records. Samples include adding products, vendors, invoices, recurring invoices, credit memos, purchase orders, receipts, and one-time checks.



Products
Search for an existing product. Enter, view and update information for existing products. Create new products.

Vendors
Search for an existing vendor. Enter, view and update information for existing vendors. Create new vendors.

Invoices
Search for an existing invoice. Enter, view and update information for existing invoices. Create new invoices.

Recurring Invoices
Search for an existing recurring invoice. Enter, view and update information for existing recurring invoices. Create new recurring invoices.

Credit Memos
Search for an existing credit memo. Enter, view and update information for existing credit memos. Create new credit memos.

Purchase Orders
Search for an existing purchase order. Enter, view and update information for existing purchase orders. Create new purchase orders.

Receipts
Search for an existing receipt. Enter, view and update information for existing receipts. Create new receipts.

Requisitions
Search for an existing requisitions. View and update information for existing requisitions.

Welcome to Accounts Payable
Done

Adding a Product Record

This sample illustrates adding an Accounts Payable product.

```
Public Function AddProduct(ByVal sProdDescription As String, ByVal sUserDefinedId As _
                   String, ByVal sUnitOfMeasure As String, ByVal bDisAllowNewlineItems As _
                   Boolean, ByVal dUnitCost As Double, ByVal lQuantityDecimal As Long, _
                            ByVal sReceivingLocation As String) As Long

    'sProdDescription = "Manila Folders"
    'sUserDefinedId = "2126"
    'sUnitOfMeasure = "Each" 'This entry should be in the Units Of Measure Table
    'bDisAllowNewlineItems = False
    'lUnitCost = 1.15
    'lQuantityDecimal = 2
    'sReceivingLocation = "Shipping - Office Supplies" 'Goes in the Locations Table

    Dim lProductId As Long

    Dim oProduct As cProduct
    Set oProduct = New cProduct

    With oProduct
        .Init goSessionContext
        .Fields(PRODUCTS_fld_DESCRIPTION) = sProdDescription
        .Fields(PRODUCTS_fld_USERDEFINEDID) = sUserDefinedId
        .Fields(PRODUCTS_fld_UNITOFMEASURE) = sUnitOfMeasure
        .Fields(PRODUCTS_fld_DISALLOWNEWLINEITEMS) = bDisAllowNewlineItems
        .Fields(PRODUCTS_fld_STANDARDUNITCOST) = dUnitCost
        .Fields(PRODUCTS_fld_QUANTITYDECIMALS) = lQuantityDecimal
        .Fields(PRODUCTS_fld_RECEIVINGLOCATION) = sReceivingLocation
        .Save
        lProductId = .Fields(PRODUCTS_fld_PRODUCTS7ID)
        .CloseDown
    End With
    AddProduct = lProductId
    Set oProduct = Nothing
End Function

'This function deletes the Product whose Id is passed to it.
Public Sub DeleteProduct(lProductId As Long)
    Dim oProduct As cProduct
    Set oProduct = New cProduct
    With oProduct
        .Init goSessionContext
        .Load lProductId
        .Delete
        .CloseDown
    End With
    Set oProduct = Nothing
End Sub
'This procedure displays the form where the Product details can be modified
Public Sub displayProductForm(ByVal lProductId As Long)
```

(Continued- page 2 of 2)

```
    Dim oProduct As cProduct
    Set oProduct = New cProduct
    With oProduct
        .Init goSessionContext
        .Load lProductId
    End With

    Dim oProductForm As cProductsForm
    Set oProductForm = New cProductsForm
    With oProductForm
        .Init goSessionContext
        Set .APProductObject = oProduct
        .ShowForm True, , True
        .CloseDown
    End With

    With oProduct
        .Save
        .CloseDown
    End With

    Set oProduct = Nothing
    Set oProductForm = Nothing
End Sub
```

Adding a Vendor Record

With this code sample, you can add a new vendor record.

```
Public Function AddVendor(ByVal sVendorName As String, _
                       ByVal sUserDefId As String, ByVal lCustNum As Long, _
                       ByVal sTaxNumber As String, ByVal sDefaultPayMethod As String, _
                       ByVal sStatus As String, ByVal sBanksDesc As String, _
                       ByVal sPaymentOption As String, Optional ByVal sTermsDesc As String,_
                       Optional ByVal dCreditLimit As Double) As Long

    'sVendorName = "Smith and Smith"
    'sUserDefId = "27"
    'lCustNum = 1999
    'sTaxNumber = "27-07-197927"
    'sDefaultPayMethod = "Check"
    'sStatus = "Active"
    'sBanksDesc = "Operating"
    'sTermsDesc = "Net 30"
    'dCreditLimit = 1000

    Dim lBanksId As Long
    'lBanksId includes the bank ID if the name of the Bank passed is valid.
    'Else it will have 0
    lBanksId = GetBanksIdFromDesc(sBanksDesc)

    Dim lTermsId As Long
    'lTermsId includes the Term ID if the name of the Term passed is valid.
    'Else it will have 0
    lTermsId = GetTermsIdFromDesc(sTermsDesc)

    Dim lVendorId As Long

    Dim oVendor As cAPVendor
    Set oVendor = New cAPVendor

    With oVendor
        .Init goSessionContext
        .Fields(APVENDORS_fld_VENDORNAME) = sVendorName
        .Fields(APVENDORS_fld_USERDEFINEDID) = sUserDefId
        .Fields(APVENDORS_fld_CUSTOMERNUMBER) = lCustNum
        .Fields(APVENDORS_fld_TAXIDNUMBER) = sTaxNumber
        .Fields(APVENDORS_fld_DEFAULTPAYMENTMETHOD) = sDefaultPayMethod
        .Fields(APVENDORS_fld_STATUS) = sStatus

        If lBanksId Then
            .Fields(APVENDORS_fld_BANKSID) = lBanksId
        End If

        If lTermsId Then
            .Fields(APVENDORS_fld_TERMSID) = lTermsId
        End If
```

(Continued- page 2 of 3)

```
        If dCreditLimit Then
            .Fields(APVENDORS_fld_HASCREDITLIMIT) = True
            .Fields(APVENDORS_fld_CREDITLIMITAMOUNT) = dCreditLimit
        End If

        .Fields(APVENDORS_fld_PAYMENTOPTION) = sPaymentOption
        .Save
        lVendorId = .Fields(APVENDORS_fld_AP7VENDORSID)
        .CloseDown
    End With
    AddVendor = lVendorId
    Set oVendor = Nothing
End Function

'Sets the distribution for the Vendor whose Id is passed to it.
Public Sub AddVendorDistribution(ByVal lVendorId As Long, ByVal sDebitAcctNum1 As String, _
                            ByVal sAcctPercent1 As String, ByVal sAcctSequence1 As String, _
                            ByVal sDebitAcctNum2 As String, ByVal sAcctPercent2 As String, _
                            ByVal sAcctSequence2 As String)
    Dim oVendor As cAPVendor
    Set oVendor = New cAPVendor

    With oVendor
        .Init goSessionContext
        .Load lVendorId

        With .Distribution.Add
            .Fields(BBDISTRIBUTIONS_fld_DEBITACCTNUM) = sDebitAcctNum1
            .Fields(BBDISTRIBUTIONS_fld_PERCENT) = sAcctPercent1
            .Fields(BBDISTRIBUTIONS_fld_SEQUENCE) = sAcctSequence1
        End With

        With .Distribution.Add
            .Fields(BBDISTRIBUTIONS_fld_DEBITACCTNUM) = sDebitAcctNum2
            .Fields(BBDISTRIBUTIONS_fld_PERCENT) = sAcctPercent2
            .Fields(BBDISTRIBUTIONS_fld_SEQUENCE) = sAcctSequence2
        End With
        .Save
        .CloseDown
    End With
    Set oVendor = Nothing
End Sub

'Displays the Vendor Form for the Vendor whose Id is passed to it.
Public Sub displayVendorForm(lVendorId As Long)
    Dim oVendor As cAPVendor
    Set oVendor = New cAPVendor
    With oVendor
        .Init goSessionContext
        .Load lVendorId
    End With
```

(Continued- page 3 of 3)

```
    Dim oVendorForm As cAPVendorsForm
    Set oVendorForm = New cAPVendorsForm

    With oVendorForm
        .Init goSessionContext
        Set .APVendorObject = oVendor
        .ShowForm True, , True
        .CloseDown
    End With

    With oVendor
        .Save
        .CloseDown
    End With

    Set oVendor = Nothing
    Set oVendorForm = Nothing
End Sub

'Deletes the vendor whose ID is passed.
Public Sub DeleteVendor(lVendorId As Long)
    Dim oVendor As cAPVendor
    Set oVendor = New cAPVendor

    With oVendor
        .Init goSessionContext
        .Load lVendorId
        .Delete
        .CloseDown
    End With
    Set oVendor = Nothing
End Sub
```

Adding an Invoice Record

With this code sample, you can add a new ***Accounts Payable*** invoice record.

```
Public Function AddInvoice(ByVal lInvoiceNumber As Long, ByVal dInvoiceAmount As Double, _
                    ByVal dtInvoiceDate As Date, ByVal dtInvoiceDueDate As Date, _
                    ByVal sInvoiceDescription As String, Optional ByVal lVendorId As Long, _
                    Optional ByVal sVendorName As String) As Long


    'sVendorName = "ADS Security Systems"
    'lInvoiceNumber = 9898
    'dInvoiceAmount = 100
    'dtInvoiceDate = Date
    'dtInvoiceDueDate = DateAdd("M", 3, dtInvoiceDate)
    'sInvoiceDescription = "Security Camera"

    Dim lInvoiceId As Long

    'If lVendorId is missing (lVendorId = 0), get the VendorId from its name.
    'If VendorName is missing, or the given VendorName is not in the list of vendors the
    'Utility function will return 0

    If lVendorId = 0 Then
        lVendorId = goFE_Service.APVendorsGetID(sVendorName, True)
        If lVendorId = 0 Then
            AddInvoice = -1
            Exit Function
        End If
    End If

    Dim oInvoice As CAPInvoice
    Set oInvoice = New CAPInvoice
    With oInvoice
        .Init goSessionContext
        .Fields(APINVOICES_fld_AP7VENDORSID) = lVendorId
        'Applies vendor default to the invoice eg. Discount Percent, Distributions. etc.
        .ApplyVendorDefaults True
        .Fields(APINVOICES_fld_INVOICENUMBER) = lInvoiceNumber
        .Fields(APINVOICES_fld_DESCRIPTION) = sInvoiceDescription
        .Fields(APINVOICES_fld_DUEDATE) = dtInvoiceDueDate
        .Fields(APINVOICES_fld_INVOICEAMOUNT) = dInvoiceAmount
        .Fields(APINVOICES_fld_INVOICEDATE) = dtInvoiceDate
        .Save
        lInvoiceId = .Fields(APINVOICES_fld_AP7INVOICESID)
        .CloseDown
    End With
    Set oInvoice = Nothing
    AddInvoice = lInvoiceId
End Function

'This function displays the Invoice Entry Form when the inovice id is passed to it.
Public Sub displayInvoiceForm(ByVal lInvoiceId As Long)
```

(Continued- page 2 of 2)

```
    Dim oInvoice As CAPInvoice
    Set oInvoice = New CAPInvoice

    With oInvoice
        .Init goSessionContext
        .Load lInvoiceId
    End With

    Dim oInvoiceForm As CAPInvoiceForm
    Set oInvoiceForm = New CAPInvoiceForm

    With oInvoiceForm
        .Init goSessionContext
        Set .APInvoiceObject = oInvoice
        .ShowForm True, , True
        .CloseDown
    End With
    Set oInvoiceForm = Nothing

    With oInvoice
        .Save
        .CloseDown
    End With
    Set oInvoice = Nothing

End Sub

'This function deletes the invoices whose ID is passed to it.
Public Sub DeleteInvoice(lInvoiceId As Long)
    Dim oInvoice As CAPInvoice
    Set oInvoice = New CAPInvoice
    With oInvoice
        .Init goSessionContext
        .Load lInvoiceId
        .Delete
        .CloseDown
    End With
    Set oInvoice = Nothing
End Sub
```

Adding a Recurring Invoice Record

With this code sample, you can add a new ***Accounts Payable*** recurring invoice.

```
Public Function AddRecurringInvoice( _
             ByVal sVendorName As String, ByVal dInvoiceAmount As Double, _
             ByVal lInvoiceNumber As Long, ByVal dtInvoiceStartDate As Date, _
             ByVal sInvoiceFrequency As String, ByVal lNumberOfInvoices As Long, _
             ByVal sRecInvoiceDescription As String, ByVal sScheduledType As String) As Long
    'lVendorName = "Bell Telephones"
    'dInvoiceAmount = 1000
    'lInvoiceNumber = 9779
    'dtInvoiceStartDate = Date
    'sInvoiceFrequency = "Monthly"
    'lNumberOfInvoices = 10
    'sRecInvoiceDescription = "Recurring Invoices"
    'sScheduledType = "Fixed"

    Dim oRecInvoice As CAPRecurringInvoice
    Set oRecInvoice = New CAPRecurringInvoice

    Dim lVendorId As Long
    lVendorId = goFE_Service.APVendorsGetID(sVendorName, True)

    Dim lRecInvoiceId As Long
    With oRecInvoice
        .Init goSessionContext
        .Fields(APRECURRINGINVOICES_fld_AP7VENDORSID) = lVendorId
        .ApplyVendorDefaults
        .Fields(APRECURRINGINVOICES_fld_INVOICENUMBER) = lInvoiceNumber
        .Fields(APRECURRINGINVOICES_fld_DESCRIPTION) = sRecInvoiceDescription
        .Fields(APRECURRINGINVOICES_fld_INVOICEAMOUNT) = dInvoiceAmount
        .Fields(APRECURRINGINVOICES_fld_SCHEDULETYPE) = sScheduledType
        .Fields(APRECURRINGINVOICES_fld_NUMBEROFINVOICES) = lNumberOfInvoices
        With .Schedule
            .Fields(SCHEDULE_fld_FREQUENCY) = sInvoiceFrequency
            .Fields(SCHEDULE_fld_STARTDATE) = dtInvoiceStartDate
            .UpdateScheduleDetails
        End With
        .UpdateScheduledInvoicesFromSchedule
        .Save
        lRecInvoiceId = Val(.Fields(APRECURRINGINVOICES_fld_AP7RECURRINGINVOICESID))
        .CloseDown
    End With

    AddRecurringInvoice = lRecInvoiceId
    Set oRecInvoice = Nothing
End Function

'Displays the Recurring Invoice Form
Public Function displayRecurringInvoice(ByVal lInvoiceId As Long)
    Dim oRecInvoice As CAPRecurringInvoice
    Set oRecInvoice = New CAPRecurringInvoice
```

(Continued- page 2 of 2)

```
    With oRecInvoice
        .Init goSessionContext
        .Load lInvoiceId
    End With

    Dim oRecInvoiceForm As CAPRecurringInvoiceForm
    Set oRecInvoiceForm = New CAPRecurringInvoiceForm

    With oRecInvoiceForm
        .Init goSessionContext
        Set .APRecurringInvoiceObject = oRecInvoice
        .ShowForm True, , True
        .CloseDown
    End With
    Set oRecInvoiceForm = Nothing

    With oRecInvoice
        .Save
        .CloseDown
    End With
    Set oRecInvoice = Nothing
End Function
```

Adding a Credit Memo Record

This code sample illustrates adding a credit memo.

```
Public Function AddCreditMemo(ByVal dtCreditMemoDate As Date, _
                              ByVal lCreditMemoNumber As Long, _
                              ByVal sCreditMemoDescription As String, _
                              ByVal dCreditMemoAmount As Double, _
                              ByVal dtPostDate As Date, ByVal sPostStatus As String, _
                              Optional ByVal lVendorId As Long, Optional ByVal sVendorName
                              As String) As Long

    'sVendorName = "Bell Telephones"
    'dtCreditMemoDate = "06/17/2002"
    'lCreditMemoNumber = 420
    'sCreditMemoDescription = "Credit Memo For Bell Telephones"
    'dCreditMemoAmount = 100
    'dtPostDate = Date
    'sPostStatus = "Not yet posted"

    Dim lCreditMemoId As Long

    If lVendorId = 0 Then
        lVendorId = goFE_Service.APVendorsGetID(sVendorName, True)
        If (lVendorId = 0) Then
            AddCreditMemo = -1
            Exit Function
        End If
    End If

    Dim oCreditMemo As CAPCreditMemo
    Set oCreditMemo = New CAPCreditMemo

    With oCreditMemo
        .Init goSessionContext
        .Fields(APCREDITMEMOS_fld_AP7VENDORSID) = lVendorId
        .ApplyVendorDefaults True
        .Fields(APCREDITMEMOS_fld_AMOUNT) = dCreditMemoAmount
        .Fields(APCREDITMEMOS_fld_CREDITMEMODATE) = dtCreditMemoDate
        .Fields(APCREDITMEMOS_fld_CREDITMEMONUMBER) = lCreditMemoNumber
        .Fields(APCREDITMEMOS_fld_DESCRIPTION) = sCreditMemoDescription
        .Fields(APCREDITMEMOS_fld_POSTDATE) = dtPostDate
        .Fields(APCREDITMEMOS_fld_POSTSTATUS) = sPostStatus
        .Save
        lCreditMemoId = .Fields(APCREDITMEMOS_fld_AP7CREDITMEMOSID)
        .CloseDown
    End With

    AddCreditMemo = lCreditMemoId
    Set oCreditMemo = Nothing
End Function
```

Adding a Purchase Order Record

With this code sample, you can add a purchase order.

```
Public Function AddPurchaseOrder(ByVal lPONumber As Long, ByVal sPOType As String, _
                                 ByVal dtPODate As Date, ByVal sPOStatus As String, _
                                 ByVal lOrderFrom As String, ByVal sShipVia As String, _
                                 ByVal sComments As String, ByVal sBuyer As String, _
                                 ByVal sDepartment As String, ByVal sConfirmTo As String, _
                                 ByVal lShipTo As String, ByVal sAttention As String, _
                                 Optional ByVal lVendorId As Long, _
                                 Optional ByVal sVendorName As String) As Long
    'sVendorName = "ADS Security Systems"
    'This Vendor Should exist in the list of vendors
    'lPONumber = 5858
    'sPOType = "Regular"
    'dtPODate = "07/17/2001"
    'sPOStatus = "Unprinted purchase order"
    'Default for new PO
    'lOrderFrom = 1
    'Primary Address of the Vendor
    'sShipVia = "FED-EX"
    'sBuyer = "Susan Thomas"
    'There should be an entry for "Susan Thomas" in the table of buyers
    'sDepartment = "Maintenance"
    'There should be an entry for "Maintenance" in the table of Departments
    'sConfirmTo = "Manager"
    'lShipTo = 1 'Primary Address
    'sAttention = "Susan Thomas"
    'sComments = "Purchase Order for ADS Security Systems"

    Dim lPurchaseOrderId As Long

    If lVendorId = 0 Then
        lVendorId = goFE_Service.APVendorsGetID(sVendorName, True)
        If lVendorId = 0 Then
        AddPurchaseOrder = -1
        Exit Function
        End If
    End If

    Dim oPurchaseOrder As CAPPurchaseOrder
    Set oPurchaseOrder = New CAPPurchaseOrder


    With oPurchaseOrder
        .Init goSessionContext
        .Fields(APPURCHASEORDERS_fld_AP7VENDORSID) = lVendorId
        .ApplyVendorDefaults
        .Fields(APPURCHASEORDERS_fld_PURCHASEORDERNUMBER) = lPONumber
        .Fields(APPURCHASEORDERS_fld_PURCHASEORDERTYPE) = sPOType
        .Fields(APPURCHASEORDERS_fld_ORDERDATE) = dtPODate
        .Fields(APPURCHASEORDERS_fld_STATUS) = sPOStatus
```

(Continued- page 2 of 3)

```
        .Fields(APPURCHASEORDERS_fld_ORDERFROMCONTACT) = lOrderFrom
        .Fields(APPURCHASEORDERS_fld_SHIPVIA) = sShipVia
        .Fields(APPURCHASEORDERS_fld_BUYER) = sBuyer
        .Fields(APPURCHASEORDERS_fld_DEPARTMENT) = sDepartment
        .Fields(APPURCHASEORDERS_fld_CONFIRMTO) = sConfirmTo
        .Fields(APPURCHASEORDERS_fld_SHIPTO) = lShipTo
        .Fields(APPURCHASEORDERS_fld_ATTENTION) = sAttention
        .Fields(APPURCHASEORDERS_fld_PURCHASEORDERCOMMENT) = sComments
        .Save
        lPurchaseOrderId = .Fields(APPURCHASEORDERS_fld_AP7PURCHASEORDERSID)
        .CloseDown
    End With
    Set oPurchaseOrder = Nothing
    AddPurchaseOrder = lPurchaseOrderId
End Function

'This function adds a LineItem to the purchase order.
Public Sub AddPOLineItem(ByVal lPurchaseOrderId As Long, ByVal sLineItemType As String, _
                         ByVal dtPostDate As Date, ByVal lQuantityOrdered As Long, _
                         ByVal dtRequiredDate As Date, ByVal dtPromisedDate As Date, _
                         ByVal sRequestedBy As String, ByVal sPostStatus As String, _
                         ByVal sProductId As String, Optional ByVal sProductName As String)

    'sLineItemType = "Regular"
    'sProductId = "FAK"
    'lQuantityOrdered = 2
    'dtRequiredDate = "09/18/2002"
    'dtPromisedDate = "09/15/2002"
    'sRequestedBy = "Bill Smith"
    'sPostStatus = "Not yet posted"
    'dtPostDate = "07/17/2002"

    Dim oPurchaseOrder As CAPPurchaseOrder
    Set oPurchaseOrder = New CAPPurchaseOrder

    Dim oProduct As cProduct
    Set oProduct = New cProduct

    'Need to load the product to get the unit cost, description and unit of measure

    oProduct.Init goSessionContext
    oProduct.Load lProductId

    With oPurchaseOrder
        .Init goSessionContext
        .Load lPurchaseOrderId
        With .LineItems.Add
            .Fields(APLINEITEMS_fld_LINEITEMTYPE) = sLineItemType
            .Fields (APLINEITEMS_fld_PRODUCTNAME) = sProductName
            .Fields(APLINEITEMS_fld_QUANTITYORDERED) = lQuantityOrdered
```

(Continued- page 3 of 3)

```
             .ApplyDefaults True
             .Fields(APLINEITEMS_fld_UNITCOST) = _
               oProduct.Fields(PRODUCTS_fld_STANDARDUNITCOST)
             .Fields(APLINEITEMS_fld_DESCRIPTION) = _
               oProduct.Fields(PRODUCTS_fld_DESCRIPTION)
             .Fields(APLINEITEMS_fld_UNITOFMEASURE) = _
               oProduct.Fields(PRODUCTS_fld_UNITOFMEASURE)
             .Fields(APLINEITEMS_fld_EXTENDEDCOST) = Round(.Fields( _
                                            APLINEITEMS_fld_UNITCOST) * lQuantityOrdered, 2)
             'Round to two decimals
             .Fields(APLINEITEMS_fld_DATEREQUIRED) = dtRequiredDate
             .Fields(APLINEITEMS_fld_DATEPROMISED) = dtPromisedDate
             .Fields(APLINEITEMS_fld_REQUESTEDBY) = sRequestedBy
             .Fields(APLINEITEMS_fld_POSTSTATUS) = sPostStatus
             .Fields(APLINEITEMS_fld_POSTDATE) = dtPostDate
         End With
         .Save
         .CloseDown
     End With
     oProduct.CloseDown
     Set oProduct = Nothing
     Set oPurchaseOrder = Nothing
End Sub

'This procedure displays the PO form with fields of PO objects whose ID is passed to it.
Public Sub displayPurchaseOrderForm(ByVal lPurchaseOrderId)
    Dim oPurchaseOrder As CAPPurchaseOrder
    Set oPurchaseOrder = New CAPPurchaseOrder
    With oPurchaseOrder
        .Init goSessionContext
        .Load lPurchaseOrderId
    End With

    Dim oPurchaseOrderForm As cAPPOForm
    Set oPurchaseOrderForm = New cAPPOForm
    With oPurchaseOrderForm
        .Init goSessionContext
        Set .APPurchaseOrderObject = oPurchaseOrder
        .ShowForm True, , True
        .CloseDown
    End With
    Set oPurchaseOrderForm = Nothing

    With oPurchaseOrder
        .Save
        .CloseDown
    End With
    Set oPurchaseOrder = Nothing

End Sub
```

Adding a Receipt Record

This code sample illustrates adding an ***Accounts Payable*** receipt.

```
Public Function AddReceipt(ByVal lPurchaseOrderId As Long, sReceivedBy As String, _
                           ByVal sDescription As String, dtReceiptDate As Date) As Long

    'sReceivedBy = "Susan Thomas"
    'sDescription = "Receipt of Purchase Order"
    'dtReceiptDate = Date

    Dim oPurchaseOrder As CAPPurchaseOrder
    Set oPurchaseOrder = New CAPPurchaseOrder

    Dim lPONumber As Long
    With oPurchaseOrder
        .Init goSessionContext
        .Load lPurchaseOrderId
        lPONumber = .Fields(APPURCHASEORDERS_fld_PURCHASEORDERNUMBER)
        .CloseDown
    End With
    Set oPurchaseOrder = Nothing

    Dim lReceiptId As Long

    Dim oReceipt As CAPReceipt
    Set oReceipt = New CAPReceipt
    With oReceipt
        .Init goSessionContext
        .Fields(APRECEIPTS_fld_PONUMBER) = lPONumber
        .Fields(APRECEIPTS_fld_RECEIPTDATE) = dtReceiptDate
        .Fields(APRECEIPTS_fld_DESCRIPTION) = sDescription
        .Fields(APRECEIPTS_fld_RECEIVEDBY) = sReceivedBy
        .Save
        lReceiptId = .Fields(APRECEIPTS_fld_AP7RECEIPTSID)
        .CloseDown
    End With
    Set oReceipt = Nothing
    AddReceipt = lReceiptId
End Function

'Adds Receipt Item to the Receipt whose Id is passed.
Public Sub AddReceiptItem(ByVal lReceiptId As Long, ByVal lPOLineNumber As Long, _
                          ByVal lQuantityReceived As Long, ByVal sPostStatus As String, _
                          ByVal dtPostDate As Date)

    'lPOLineNumber = 1
    'lQuantityReceived = 1
    'sPostStatus = "Not yet posted"
    'dtPostDate = "07/19/2002"

    Dim oReceipt As CAPReceipt
    Set oReceipt = New CAPReceipt
```

(Continued- page 2 of 2)

```
    With oReceipt
        .Init goSessionContext
        .Load lReceiptId
        With .ReceiptItems.Add
            .Fields(APRECEIPTITEMS_fld_POLINENUMBER) = lPOLineNumber
            .Fields(APRECEIPTITEMS_fld_QUANTITYRECEIVED) = lQuantityReceived
            .Fields(APRECEIPTITEMS_fld_POSTSTATUS) = sPostStatus
            .Fields(APRECEIPTITEMS_fld_POSTDATE) = dtPostDate
        End With
        .Save
        .CloseDown
    End With
End Sub

'This function displays the Receipt form for the receipt whose Id is passed to it.
Public Sub displayReceiptForm(ByVal lReceiptId As Long)
    Dim oReceipt As CAPReceipt
    Set oReceipt = New CAPReceipt
    With oReceipt
        .Init goSessionContext
        .Load lReceiptId
    End With

    Dim oReceiptForm As CAPReceiptForm
    Set oReceiptForm = New CAPReceiptForm
    With oReceiptForm
        .Init goSessionContext
        Set .APReceiptObject = oReceipt
        .ShowForm True, , True
        .CloseDown
    End With
    Set oReceiptForm = Nothing

    With oReceipt
        .Save
        .CloseDown
    End With
    Set oReceipt = Nothing
End Sub
```

Adding a One-Time Check

With this code sample, you can create a one-time ***Accounts Payable*** check.

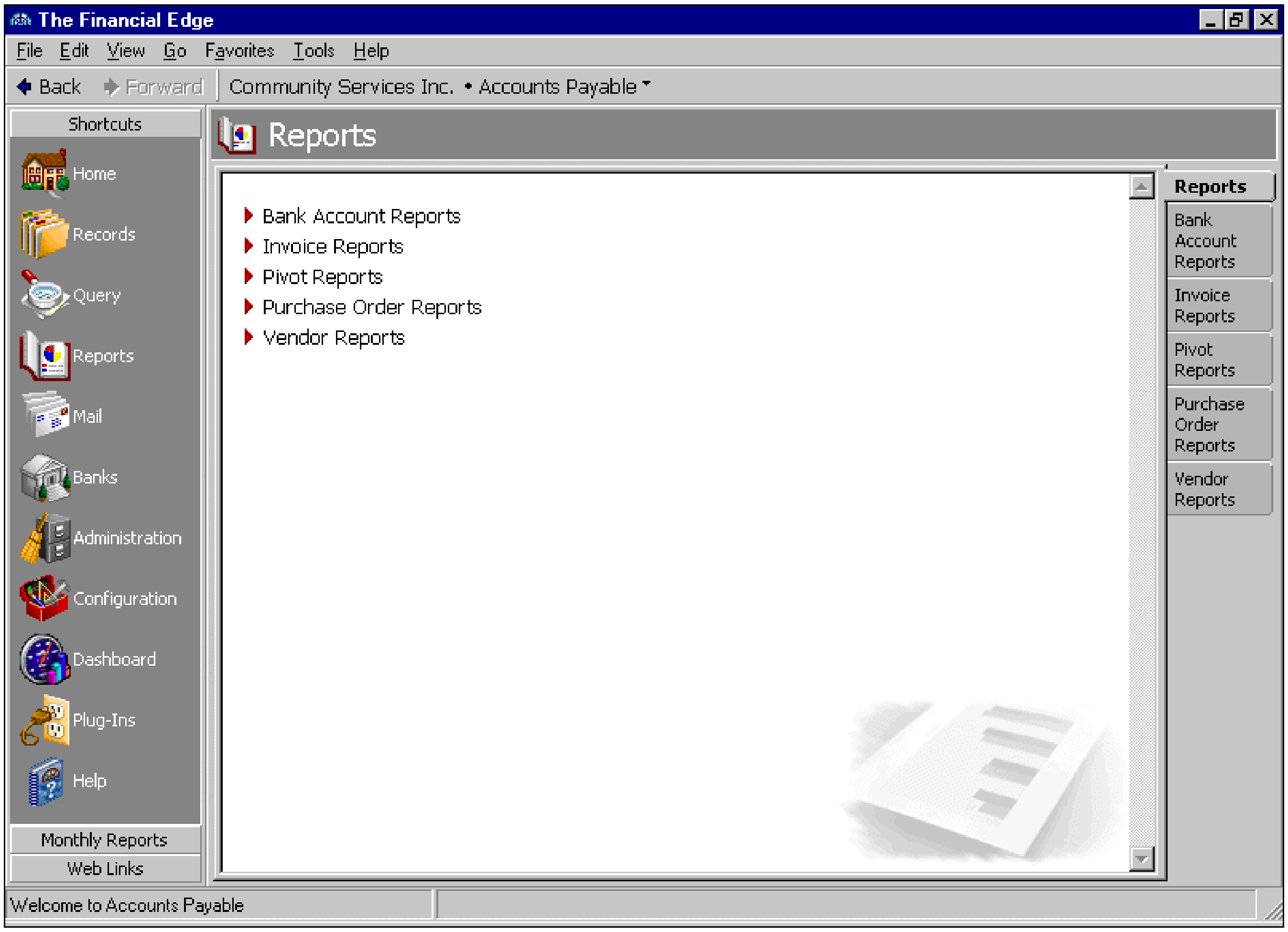
```
Public Function AddCheck() As Long
    On Error GoTo errH
    Dim lCheckId As Long
    Dim oCheck As CAPCheck
    Set oCheck = New CAPCheck

    With oCheck
        .Init goSessionContext
        'Set what type of check is being created
        .CheckType = staticentry_CheckTypes_OneTimeCheck
        'Backend ID of the Account
        .Fields(CHECKS_fld_BANKSID) = 1
        .Fields(CHECKS_fld_AMOUNT) = 35
        .StaticEntryField(CHECKS_fld_CHECKFORMAT) = _
          staticentry_PaymentFormat_CHKAPUS2LPP001
        'Another way to set the check format
        '.Fields(CHECKS_fld_CHECKFORMAT) = "CHKAPUS2LPP001"
        'Another way to set the check format
        '.Fields(CHECKS_fld_CHECKFORMAT) = _
                        goCodeTablesServer.StaticTableTranslation(staticnumPaymentFormat, _
                        staticentry_PaymentFormat_CHKAPUS1LPP001)
        .Fields(CHECKS_fld_CHECKNUMBER) = 191917
        .Fields(CHECKS_fld_PAYEENAME) = "John Doe"
        .Fields(CHECKS_fld_PRINTERFORCHECKS) = "\\your_server\printer_name"
        .Fields(CHECKS_fld_PRINTLATER) = True
        'Add Distribution information
        With .Distribution.Add
            .Fields(BBDISTRIBUTIONS_fld_DEBITACCTNUM) = "01-1000-00"
            .Fields(BBDISTRIBUTIONS_fld_AMOUNT) = 35
            .Fields(BBDISTRIBUTIONS_fld_CREDITACCTNUM) = "01-2000-00"
        End With
        'Save the check
        .Save
        lCheckId = .Fields(CHECKS_fld_CHECKS7ID)
        .CloseDown
    End With
    Set oCheck = Nothing
    'Return the checkid
    AddCheck = lCheckId
    Exit Function
errH:
    MsgBox Err.Description
    If Not oCheck Is Nothing Then
        oCheck.CloseDown
        Set oCheck = Nothing
    End If
End Function
```

(Continued- page 2 of 2)

```
'Loads the check and displays the CheckForm
Public Function DisplayCheckForm(ByVal lCheckId As Long)
    Dim oCheck As CAPCheck
    Set oCheck = New CAPCheck
    With oCheck
        .Init goSessionContext
        .Load lCheckId
    End With
    Dim oCheckForm As cCheckForm
    Set oCheckForm = New cCheckForm
    With oCheckForm
        .Init goSessionContext
        Set .CheckObject = oCheck
        .ShowForm True, , False
        'This also closes the oCheck object
        .CloseDown
    End With
    Set oCheckForm = Nothing
    Set oCheck = Nothing
End Function
```

Accounts Payable Reports Samples

This section contains code samples for creating applications you can use with ***Accounts Payable*** reports. Samples include creating vendor activity, open invoice, bank reconciliation, and purchase order detail reports.



Creating a Vendor Activity Report

With this code sample, you can create a vendor activity report.

```
Public Sub Create_Vendor_Activity_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_AP_VendorActivityReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Vendor Activity Report From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData
        'Include Transactions having 'Transaction Date' in the following range of dates
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                           bbrep_VendorActivity_DateCombo_InvoiceDate)
                           .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_SPECIFICRANGE

        'Set the Start Date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                      bbrep_VendorActivity_StartDate_InvoiceDate) _
                      .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("01/20/2000")

        'Set the End Date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                      bbrep_VendorActivity_EndDate_InvoiceDate) _
                      .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("07/20/2000")

        'Include Transactions having 'Post Date' in the last Fiscal Year
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                        bbrep_VendorActivity_DateCombo_DueDate) _
                        .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_LAST_FISCAL_YEAR

        'Show Unapplied balances for the Credit Memos
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_VendorActivity_IncludeOnlyAppliedCM) _
                            .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        'Do not include vendors with no activity in the specified range
```

(Continued- page 2 of 2)

```
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_VendorActivity_IncludeNoTransactions) _
                            .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = False

        'Set the various Filters
        'Set the Invoice Filter
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,
_
                                 valuenumber:=1, ValueSet:=CStr(bbFilterType_AP_Invoices))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Two ways to set the invoice number. Shown Below.
            'Pass the account number directly into the 'FROMVALUE' field
            .Fields(FILTERS_fld_FROMVALUE) = "34324-001"
            'Pass the Id of the Invoice number into the 'TOID' field
            .Fields(FILTERS_fld_TOID) = getInvoiceId("34324-025")
        End With

        'Set the Vendor Attribute Filter
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,
_
                          valuenumber:=1, ValueSet:=CStr(bbFilterType_AP_VendorAttributes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Set the Attribute Type to AP Vendor Type
            .Fields(FILTERS_fld_GLOBALATTRIBUTETYPE) = bbGlobalAttributeType_APVendor
            'Set the name of the Attribute on which to Filter
            .Fields(FILTERS_fld_FROMID) = getAttributeTypeId( _
                          "Solicit for Donations", bbGlobalAttributeType_APVendor)
            'Set the value of the Attribute
            .Fields(FILTERS_fld_FROMVALUE) = True
        End With

        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_AP_ReceiptPostStatus))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMVALUE) = "Not yet posted"
        End With

    End With
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating an Open Invoice Report

With this code sample, you can create an open invoice report.

```
Public Sub Create_Open_Invoice_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_AP_OpenInvoiceReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Open Invoice Report Created from API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With
    With oMetaData

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_OpenInvoice_ReportFormat) _
                            .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "Summary"

        'Show Invoices Open as of 'today'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_OpenInvoice_DateCombo_ShowInvoices) _
                            .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_TODAY

        'Invoice open date is based on 'post date'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_OpenInvoice_BaseOpenDateOn) _
                            .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "Post Date"

        'Discounts are calculated based on a specific date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_OpenInvoice_DateCombo_CalcDiscounts) _
                            .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_SPECIFICDATE

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                       bbrep_OpenInvoice_StartDate_CalcDiscounts) _
                       .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("1/20/2001")
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                       bbrep_OpenInvoice_EndDate_CalcDiscounts) _
```

(Continued- page 2 of 3)

```
                        .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("6/20/2001")

        'Base invoice aging on Post Date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_OpenInvoice_BaseAgingOn) _
                            .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "Due Date"
        'Include Transactions with these dates -
        'Transaction date between the following range of dates
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                        bbrep_OpenInvoice_DateCombo_InvoiceDate) _
                        .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_LAST_FISCAL_YEAR

        'Due date in the last fiscal year
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                        bbrep_OpenInvoice_DateCombo_DueDate) _
                        .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_LAST_FISCAL_YEAR

        'Include All Post Dates
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
                            bbrep_OpenInvoice_DateCombo_PostDate) _
                            .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        'Set the various Filters

        'Set the Vendor Filter. Include vendors in the following 2 ranges
        'valuenumber:=1 for first range
        With .PropertyDataObject(bbrep_Offset_Filters, _
                                 bbrep_FilterParameter_FilterValues, valuenumber:=1, _
                                 ValueSet:=CStr(bbFilterType_AP_Vendors)) _
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Select the start of the range by setting the name of the vendor
            .Fields(FILTERS_fld_FROMID) = goFE_Service.APVendorsGetID( _
                                                 "ADS Security Systems", True)
            'Select the end of the range by setting the vendorid
            .Fields(FILTERS_fld_TOID) = goFE_Service.APVendorsGetID( _
                                               "Main Course Catering", True)
        End With
        'valuenumber:=2 for the second range
        With .PropertyDataObject(bbrep_Offset_Filters, _
                                 bbrep_FilterParameter_FilterValues, valuenumber:=2, _
                                 ValueSet:=CStr(bbFilterType_AP_Vendors))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Select the start of the range by setting the name of the vendor
            .Fields(FILTERS_fld_FROMID) = goFE_Service.APVendorsGetID( _
                                                 "Stevenson Travel", True)
            'Select the end of the range by setting the vendorid
            .Fields(FILTERS_fld_TOID) = goFE_Service.APVendorsGetID( _
                                               "Twin Bridges Printing", True)
        End With
```

(Continued- page 3 of 3)

```
        'Set the Bank Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
                                 bbrep_FilterParameter_FilterValues, valuenumber:=2, _
                                 ValueSet:=CStr(bbFilterType_Banks))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = goFE_Service.BanksGetBankID("Operating")
        End With
    End With
    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Creating a Bank Reconciliation Report

This code sample illustrates creating a bank reconciliation report.

```
Public Sub Create_Bank_Reconciliation_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_BR_BankReconciliationReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Bank Reconciliation Report From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With

    With oMetaData





        'Set the reconciliation date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_RecDate) _
                      .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("12/31/2001")

        'Set the beginning balance
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_StartBal) _
                      .Fields(REPORTPARAMETERVALUES_fld_CURRENCY) = 300000

        'Set the ending balance
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_EndBal) _
                      .Fields(REPORTPARAMETERVALUES_fld_CURRENCY) = 600000

        'Show unreconciled transactions between the following range of dates
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_DateCombo) _
                      .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_SPECIFICDATE

        'Set the start date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_StartDate) _
                      .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("01/01/2001")
```

(Continued- page 2 of 2)

```
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_StartDate) _
                      .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("01/01/2001")

        'Set the end date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BankRec_EndDate) _
                      .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("12/31/2001")

    End With
    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating a Purchase Order Detail Report

With this code sample, you can create purchase order detail reports.

```
Public Sub Create_PODetail_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_AP_PurchaseOrderDetailReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "PO Detail Report Created From API"
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields Set by the API"
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With
    With oMetaData
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_POdetail_PODateType) _
                            .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_PODetail_ProDateType).Fields( _
          REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_PODetail_ReqDateType)
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        'Sets the PO Type Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_AP_POPOType)) _
          .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = staticentry_APPurchaseorderType_Regular
        End With

        'Sets the PO Status Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_AP_POStatus))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = staticentry_APPurchaseOrderStatus_Closed
        End With

        'Sets the Vendor Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_AP_Vendors))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
```

(Continued- page 2 of 4)

```
            .Fields(FILTERS_fld_FROMID) = goFE_Service.APVendorsGetID( _
              "ADS Security Systems", True)
            .Fields(FILTERS_fld_TOID) = goFE_Service.APVendorsGetID("Auto Express", True)
        End With

        'Sets the PurchaseOrder Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_AP_PurchaseOrders))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = _
              goFE_Service.APPurchaseOrdersGetPurchaseOrderID( _
              "5858", goFE_Service.APVendorsGetID("ADS Security Systems", True))
        End With

        'Sets the Department Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_AP_PODepartment))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getDepartmentId("Development")
        End With

        'Sets the Buyers Filter
        With .PropertyDataObject(bbrep_Offset_Filters,
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_AP_POBuyer))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getBuyerId("Tom Johnson")
        End With

        'Formatting of the Report

        'Set the Heading of the Report
        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_PrintOrgName) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_Title) _
          .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "PO Details Report of ABC Company"

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_SubTitle) _
          .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "Purchase Order Details"

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_Align) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = AlignmentConstants.vbCenter

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_PrintDate) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
```

(Continued- page 3 of 4)

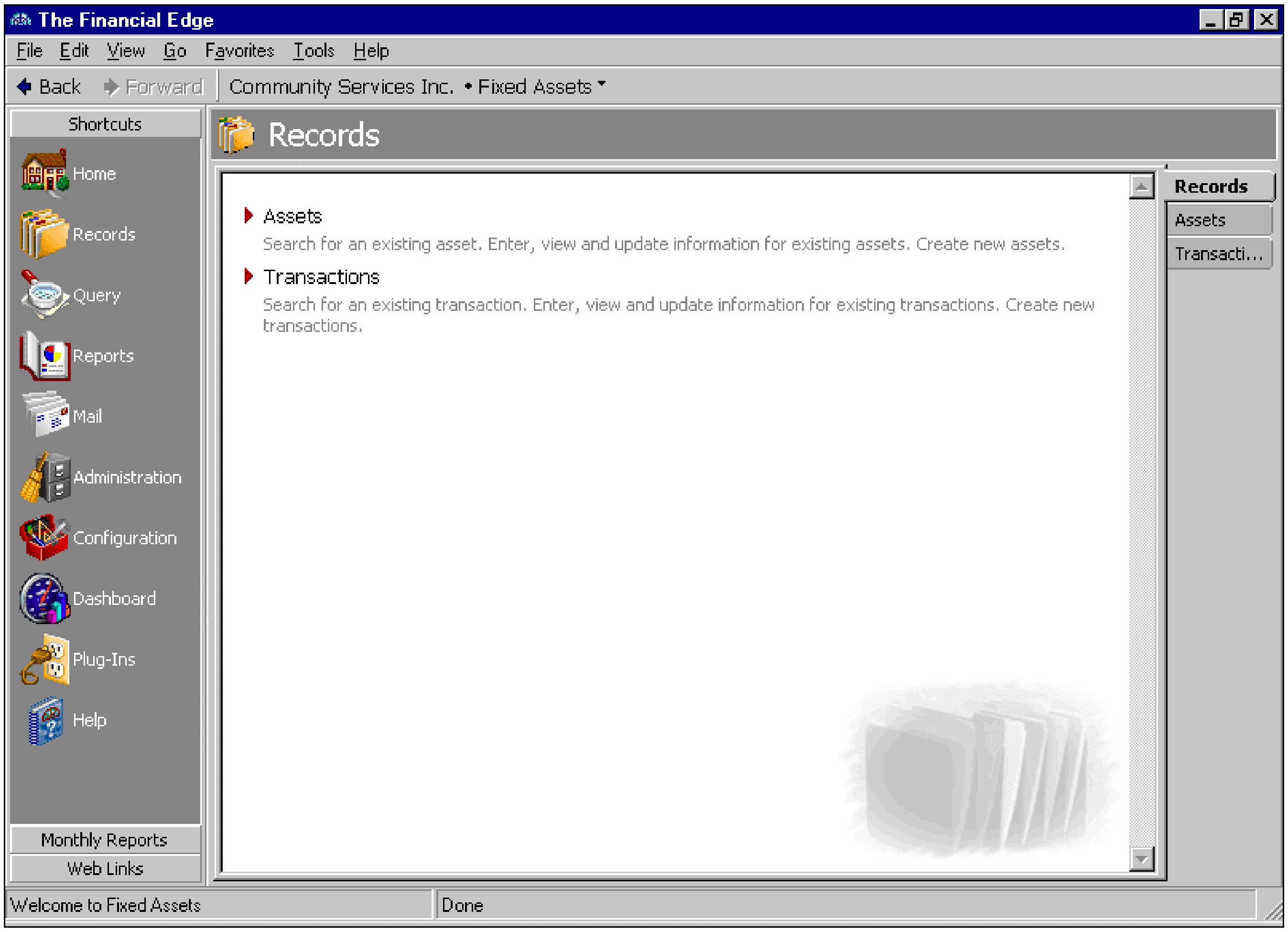
```
        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_DateAlign) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = AlignmentConstants.vbCenter

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_PrintPageNumber) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_PageNumberAlign) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = AlignmentConstants.vbCenter

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Header_PrintOnEachPage) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = False

        'Set the Format Criteria of The Report
        .PropertyDataObject(bbrep_OffSet_Format, _
          bbrep_Format_Criteria_PrintOnSeparatePage)_
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = False

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Criteria_PrintCriteria) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True

        'Apply Color Scheme to the Report
        .PropertyDataObject(bbrep_OffSet_Format, _
          bbrep_Format_ColorScheme_ApplyColorScheme)_
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True

        .PropertyDataObject( _
          bbrep_OffSet_Format, bbrep_Format_ColorScheme_ColumnHeadingBackColor) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = vbBlue

        .PropertyDataObject(bbrep_OffSet_Format, _
          bbrep_Format_ColorScheme_ColumnHeadingForeColor) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = vbYellow

        .PropertyDataObject(bbrep_OffSet_Format,
          bbrep_Format_ColorScheme_GroupHeadingBackColor) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = vbRed

        .PropertyDataObject(bbrep_OffSet_Format, _
          bbrep_Format_ColorScheme_GroupHeadingForeColor) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = vbBlack

        'Set the Fields in the Misc Tab
        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Misc_ShowCurrencyOn) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1

        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Misc_ShowPercentOn) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 2
```

(Continued- page 4 of 4)

```
        .PropertyDataObject(bbrep_OffSet_Format, bbrep_Format_Misc_NumDecimalsAmount) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 2

    End With
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Fixed Assets Records Samples

This section contains code samples for creating applications you can use with ***Fixed Assets*** records. Samples include adding assets and transactions.



Assets
Search for an existing asset. Enter, view and update information for existing assets. Create new assets.

Transactions
Search for an existing transaction. Enter, view and update information for existing transactions. Create new transactions.

Welcome to Fixed Assets
Done

Adding an Asset Record

With the following code sample, you can create a new asset record.

```
Friend Function AddAsset(ByVal lAssetUserId As Long, ByVal sDescription As String, _
                         ByVal sModelNumber As String, ByVal sSerialNumber As String, _
                         ByVal sClass As String, ByVal sLocation As String, _
                         ByVal dtAcquisitionDate As Date, ByVal dtDateInService As Date, _
                         ByVal dAcquisitionValue As Double) As Long

    'lAssetUserId = 99
    'sDescription = "Honda Accord"
    'sModelNumber = "1999 EX"
    'sSerialNumber = "CANBEVINNUMBER"
    'sClass = "Automobiles, Taxis"
    'sLocation = "Garage"
    'dtAcquisitionDate = "07/27/1999"
    'dtDateInService = "08/15/1999"
    'dAcquisitionValue = 20500

    Dim lAssetId As Long

    Dim oAsset As CFAAsset
    Set oAsset = New CFAAsset

    With oAsset
        .Init goSessionContext
        .Fields(FAASSETS_fld_USERDEFINEDID) = lAssetUserId
        .Fields(FAASSETS_fld_DESCRIPTION) = sDescription
        .Fields(FAASSETS_fld_MODELNUM) = sModelNumber
        .Fields(FAASSETS_fld_SERIALNUM) = sSerialNumber
        .Fields(FAASSETS_fld_ASSETCLASSESDESCRIPTON) = sClass
        .Fields(FAASSETS_fld_LOCATION) = sLocation
        .Fields(FAASSETS_fld_ACQUISITIONDATE) = dtAcquisitionDate
        .Fields(FAASSETS_fld_DATEINSERVICE) = dtDateInService
        .Fields(FAASSETS_fld_ACQUISITIONVALUE) = dAcquisitionValue
        .ApplyAssetClassDefaults True
        .Save
        lAssetId = .Fields(FAASSETS_fld_FAASSETSID)
        .CloseDown
    End With
    Set oAsset = Nothing
    AddAsset = lAssetId
End Function

'Displays the Asset Form for the Asset whose Id is passed to it
Friend Sub displayAssetForm(ByVal lAssetId As Long)
    Dim oAsset As CFAAsset
    Set oAsset = New CFAAsset
With oAsset
        .Init goSessionContext
        .Load lAssetId
End With
```

(Continued- page 2 of 2)

```
    Dim oAssetForm As cFAAssetForm
    Set oAssetForm = New cFAAssetForm
    With oAssetForm
        .Init goSessionContext
        Set .DataObject = oAsset
        .ShowForm True, , True
        .CloseDown
    End With
    Set oAssetForm = Nothing

    With oAsset
        .Save
        .CloseDown
    End With
    Set oAsset = Nothing

End Sub
```

Adding a Transaction Record

With this code sample, you can add a new ***Fixed Assets*** transaction.

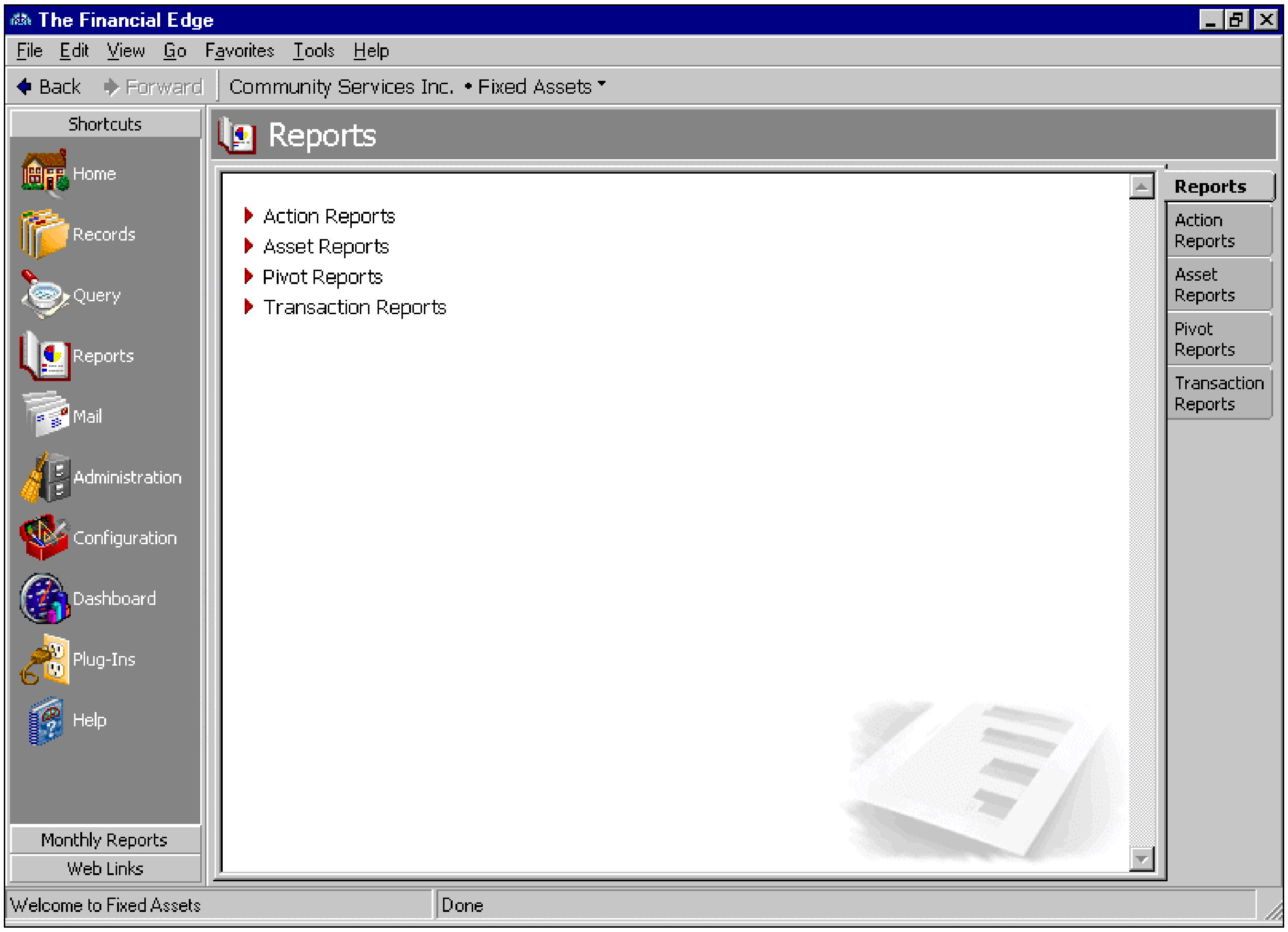
```
Friend Function AddTransaction(ByVal lAssetId As Long, ByVal sTransactiontype As String, _
                               ByVal sPostStatus As String, ByVal dtPostDate As Date, _
                               ByVal sComments As String, ByVal dtTranDate As Date, _
                               ByVal dAmount As Double) As Long

    'sTransactiontype = "Asset Acquisition"
    'sPostStatus = "Not yet posted"
    'dtPostDate = "07/27/2000"
    'sComments = "Comments on the acquisition of Honda Accord"
    'dtTranDate = "08/01/1999"
    'dAmount = 20500

    Dim lTransactionId As Long

    Dim oTransaction As CFATransaction
    Set oTransaction = New CFATransaction

    With oTransaction
        .Init goSessionContext
        .Fields(FATRANSACTIONS_fld_TRANTYPE) = sTransactiontype
        .Fields(FATRANSACTIONS_fld_FAASSETSID) = lAssetId
        .ApplyAssetDefaults True
        .Fields(FATRANSACTIONS_fld_TRANDATE) = dtTranDate
        .Fields(FATRANSACTIONS_fld_AMOUNT) = dAmount
        .Fields(FATRANSACTIONS_fld_POSTSTATUS) = sPostStatus
        .Fields(FATRANSACTIONS_fld_POSTDATE) = dtPostDate
        .Fields(FATRANSACTIONS_fld_COMMENTS) = sComments
        .Save
        lTransactionId = .Fields(FATRANSACTIONS_fld_FATRANSACTIONSID)
        .CloseDown
    End With
    Set oTransaction = Nothing
    AddTransaction = lTransactionId
End Function

'Displays the Transaction Form of the Transacation whose Id is passed to it
Public Sub displayTransactionForm(ByVal lTransactionId As Long)
    Dim oTransaction As CFATransaction
    Set oTransaction = New CFATransaction
    With oTransaction
        .Init goSessionContext
        .Load lTransactionId
    End With

    Dim oTransactionForm As CFATransactionForm
    Set oTransactionForm = New CFATransactionForm
    With oTransactionForm
        .Init goSessionContext
        Set .DataObject = oTransaction
```

(Continued- page 2 of 2)

```
         .ShowForm True, , True
         .CloseDown
     End With
     Set oTransactionForm = Nothing

     With oTransaction
         .Save
         .CloseDown
     End With
     Set oTransaction = Nothing
End Sub
```

Fixed Assets Reports Samples

This section contains code samples for creating applications you can use with ***Fixed Assets*** reports. Samples include creating book value, action listing, and depreciation summary reports.



Creating a Book Value Report

With this code sample, you can create a book value report.

```
Public Sub Create_Book_Value_Reports()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_FA_BookValueReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Book Value Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData

        'Calculate Book Value as of 'Today'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BookValue_AsOfDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = bbDATE_TODAY

        'Include Assets which have service date in the following range
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_BookValue_InServiceDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_SPECIFICRANGE
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_BookValue_InServiceStartDate) _
          .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("01/01/2001")
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_BookValue_InServiceEndDate) _
          .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("06/01/2001")

        'Exclude Disposed Assets
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, bbrep_BookValue_ExcludeDisposed) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        'Set the Filters

        'Set the Depreciation Method Filter and select Methods to include in the report
        'Select the First Method - ACRS
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,_
          valuenumber:=1, ValueSet:=CStr(bbFilterType_FA_DepreciationMethods))
```

(Continued- page 2 of 3)

```
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_CATEGORY) = staticentry_FADepreciationMethod_ACRS
        End With

        'Select the Second Method - Declining Balance
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=2, _
          ValueSet:=CStr(bbFilterType_FA_DepreciationMethods))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_CATEGORY) = _
              staticentry_FADepreciationMethod_DecliningBalance
        End With

        'Set the Department Filter
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,_
             valuenumber:=1, ValueSet:=CStr(bbFilterType_FA_Departments))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = searchInCodeTable("Department", "Development")
        End With

'Set the Disposal Method Filter - Filter by Exchange Disposal Method
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_FA_DisposalMethods))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getIdFromCodeTable( _
              ctnumFADisposalMethod, "Exchange")
        End With

        'Set the Asset Attribute Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_FA_AssetAttributes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            'Set the Attribute Type to AP Vendor Type
            .Fields(FILTERS_fld_GLOBALATTRIBUTETYPE) = bbGlobalAttributeType_FAAsset
            'Set the name of the Attribute on which to Filter
            .Fields(FILTERS_fld_FROMID) = getAttributeTypeId( _
              "Assigned to", bbGlobalAttributeType_FAAsset)
            'Set the value of the Attribute
            .Fields(FILTERS_fld_FROMVALUE) = "Sandy Johnson"
        End With

        'Set the Location Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
```

(Continued- page 3 of 3)

```
          valuenumber:=1, ValueSet:=CStr(bbFilterType_FA_Locations))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = searchInCodeTable( _
                                                "Location", "Information Technology")
        End With

        'Set the Classes Filter
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_FA_Classes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getAssetClassId("Buildings")
        End With

    End With

    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating an Action Listing Report

With this code sample, you can create an action listing report.

```
Public Sub Create_Action_Listing_Report()
    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_FA_ActionListing)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Action Listing Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the api"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With

    With oMetaData

        'Include Actions with the 'In Service' date as 'Today'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_InServiceDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_TODAY

        'Include Actions with the 'Disposal' date between the range of dates specified
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_DisposalDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_SPECIFICRANGE
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_DisposalStartDate) _
          .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("01/01/2001")
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_DisposalEndDate) _
          .Fields(REPORTPARAMETERVALUES_fld_DATETIME) = DateValue("06/01/2001")

        'Include Actions with any 'Action' Date
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_ActionDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        'Include Actions with High and Normal Priority and Exclude Actions with Low Priority
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
```

(Continued- page 2 of 2)

```
          bbrep_ActionListing_IncludeHighPriority) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_IncludeNormalPriority) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_IncludeLowPriority) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = False

        'Include Incomplete Actions, Exclude Complete Actions
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_IncludeIncomplete) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_ActionListing_IncludeComplete) _
          .Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = False

        'Set the Filters

        'Set the Action Type Filter to 'Maintenance'
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_ActionTypes))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getIdFromCodeTable(ctnumFAActionType, _
              "Maintenance")
        End With

        'Set the Action Statuses Filter to 'In Progress'
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, _
          valuenumber:=1, ValueSet:=CStr(bbFilterType_ActionStatuses))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getIdFromCodeTable(ctnumFAActionStatus, _
              "In progress")
        End With

    End With

    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Creating a Depreciation Summary Report

With this code sample, you can create a depreciation summary report.

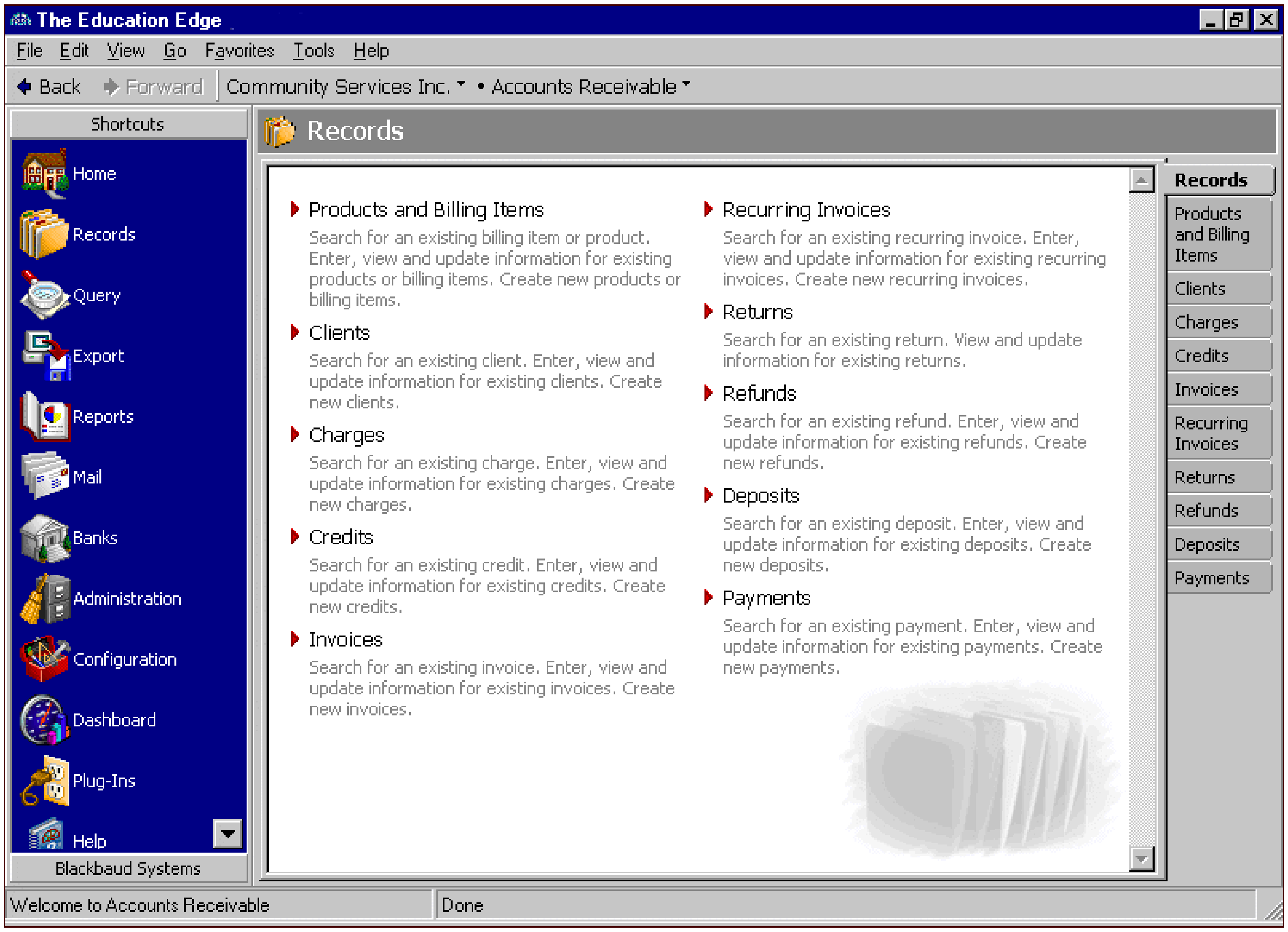
```
Public Sub Create_Depreciation_Summary_Report()

    Dim oReport As IBBReportInstance
    Set oReport = goFE_Service.CreateReportInstance(bbrep_FA_DepreciationSummaryReport)

    Dim oMetaData As IBBReportMetaData
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Depreciation Summary Report from API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False

    End With

    With oMetaData

        'Include Assets with the 'In Service' date in 'Last Calendar Year'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific,
          bbrep_DepreciationSummary_InServiceDateType).Fields( _
          REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_LASTYEAR

        'Include Assets with the 'Disposal' date in 'This Calendar Year'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepreciationSummary_InServiceDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_THISYEAR

        'Include Depreciation Transactions with date in the 'Last Calendar Year'
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepreciationSummary_TransactionDateType) _
          .Fields(REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_LASTYEAR

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepreciationSummary_SummarizeBy) _
          .Fields(REPORTPARAMETERVALUES_fld_TEXT) = "Location" 'Constant ?

        'Set the Filters
        'Set the Depreciation Method Filter - MACRS
```

(Continued- page 2 of 2)

```
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_FA_DepreciationMethods))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_CATEGORY) = staticentry_FADepreciationMethod_MACRS
        End With

        'Set the Disposal Method Filter - Filter by Exchange Disposal Method
        With .PropertyDataObject(bbrep_Offset_Filters, _
          bbrep_FilterParameter_FilterValues, valuenumber:=1, _
          ValueSet:=CStr(bbFilterType_FA_DisposalMethods))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Selected
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = getIdFromCodeTable(ctnumFADisposalMethod, _
              "Retirement")
        End With

    End With

    'Show the Parameter Form and Close the Report
    With oReport
        .Process bbrep_ProcessOption_ShowParameterForm, True
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Accounts Receivable Records Samples

This section contains code samples for creating applications you can use with ***Accounts Receivable*** records. Samples include adding products and billing items, clients, charges, credits, invoices, recurring invoices, returns, refunds, deposits, and payments.



Products and Billing Items

Search for an existing billing item or product. Enter, view and update information for existing products or billing items. Create new products or billing items.

Clients

Search for an existing client. Enter, view and update information for existing clients. Create new clients.

Charges

Search for an existing charge. Enter, view and update information for existing charges. Create new charges.

Credits

Search for an existing credit. Enter, view and update information for existing credits. Create new credits.

Invoices

Search for an existing invoice. Enter, view and update information for existing invoices. Create new invoices.

Recurring Invoices

Search for an existing recurring invoice. Enter, view and update information for existing recurring invoices. Create new recurring invoices.

Returns

Search for an existing return. View and update information for existing returns.

Refunds

Search for an existing refund. Enter, view and update information for existing refunds. Create new refunds.

Deposits

Search for an existing deposit. Enter, view and update information for existing deposits. Create new deposits.

Payments

Search for an existing payment. Enter, view and update information for existing payments. Create new payments.

Adding a Client Record

With the following code sample, you can create a new client record.

```
Private Sub Add_ARClient(ByVal lClientRecordType As EST_ARClientRecordType, _
ByVal sClientName As String, ByVal sClientID As String, ByVal sStatus As String, _
ByVal sClientType As String, ByVal sProvider As String, ByVal sUserID As String, _
ByVal sPIN As String, ByVal sCFDA As String, ByVal sIndustry As String, _
ByVal sTerritory As String, ByVal sGender As String, ByVal sSSN As String, _
ByVal dtBirthDate As Date, ByVal sReligion As String, ByVal sEthnicity As String)

'    lClientRecordType = staticentry_ARClientRecordType_Individual Or lClientRecordType = _
          'staticentry_ARClientRecordType_Organization
'    sClientName = "Client 1"
'    sClientID = "1"
'    sStatus = "Active"
'    sClientType = "Social Club"
'    sProvider = "Lois"
'    sUserID = "CL1"
'    sPIN = "4489"
'    sGender = "Male"
'    sSSN = "123-45-6789"
'    dtBirthDate = "09/12/1973"
'    sReligion = "Baptist"
'    sEthnicity = "Caucasian"
'    sCFDA = "CFDA 1"
'    sIndustry = "Retail"
'    sTerritory = "Green"    Dim lAssetId As Long

Dim oClient As CARClient
    Dim oWarningRule As IBBWarningRule
    Set oClient = New CARClient

    With oClient
        .Init goSessionContext
        .StaticEntryField(ARCLIENTS_fld_RECORDTYPE) = lClientRecordType
        .Fields(ARCLIENTS_fld_CLIENTNAME) = sClientName
        .Fields(ARCLIENTS_fld_USERDEFINEDID) = sClientID
        .Fields(ARCLIENTS_fld_ACCOUNTSTATUS) = sStatus
        .Fields(ARCLIENTS_fld_CLIENTTYPE) = sClientType
        .Fields(ARCLIENTS_fld_SERVICEPROVIDER) = sProvider
        .Fields(ARCLIENTS_fld_CLIENTUSERID) = sUserID
        .Fields(ARCLIENTS_fld_CLIENTPIN) = sPIN

        If lClientRecordType = staticentry_ARClientRecordType_Individual Then
            'Client type is Individual
            With .IndividualNameObject
                .Fields(NAME_fld_GENDER) = sGender
                .Fields(NAME_fld_SSN) = sSSN
                .Fields(NAME_fld_BIRTHDATE) = dtBirthDate
                .Fields(NAME_fld_RELIGION) = sReligion
                .Fields(NAME_fld_ETHNICITY) = sEthnicity
```

(Continued- page 2 of 2)

```
            End With

        Else    'Client type is Organizaton
            .Fields(ARCLIENTS_fld_CFDANUMBER) = sCFDA
            .Fields(ARCLIENTS_fld_INDUSTRY) = sIndustry
            .Fields(ARCLIENTS_fld_TERRITORY) = sTerritory
        End If

'This needs to be done if you want to override warning rule given regarding the _
        'primary address if one does not exist
        Set oWarningRule = oClient
        oWarningRule.OverrideWarning(ARClient_Warning_PrimaryAddress) = True

        .Save
        .CloseDown
    End With

    Set oClient = Nothing
    Set oWarningRule = Nothing

End Sub
```

Adding an Accounts Receivable Invoice Record

With the following code sample, you can create a new invoice record.

```
Private Sub Add_ARInvoice(ByVal lClientID As Long, ByVal dtInvoiceDate As Date, _
ByVal bOnHold As Boolean, ByVal sTermsDiscountPercent As String, _
ByVal dtTermsDiscountDate As Date, ByVal dtDueDate As Date, ByVal sOrderBy As String, _
ByVal dtOrderDate As Date, ByVal lBillToAddressID As Long, ByVal lShipToAddressID As Long)

'    lClientID = 1
'    dtInvoiceDate = "09/23/2004"
'    bOnHold = False
'    sTermsDiscountPercent = 1
'    dtTermsDiscountDate = "09/30/2004"
'    dtDueDate = "10/23/2004"
'    sOrderBy = "ABC"
'    dtOrderDate = "09/23/2004"
'    lBillToAddressID = 1
'    lShipToAddressID = 1

    Dim oARInvoice As CARInvoice
    Set oARInvoice = New CARInvoice

    With oARInvoice
        .Init goSessionContext
        .Fields(ARINVOICES_fld_AR7CLIENTSID) = lClientID
        .Fields(ARINVOICES_fld_INVOICEDATE) = dtInvoiceDate
        .Fields(ARINVOICES_fld_ONHOLD) = bOnHold
        .Fields(ARINVOICES_fld_TERMSDISCOUNTPERCENT) = sTermsDiscountPercent
        .Fields(ARINVOICES_fld_TERMSDISCOUNTDATE) = dtTermsDiscountDate
        .Fields(ARINVOICES_fld_DUEDATE) = dtDueDate
        .Fields(ARINVOICES_fld_ORDEREDBY) = sOrderBy
        .Fields(ARINVOICES_fld_ORDEREDONDATE) = dtOrderDate
        .Fields(ARINVOICES_fld_BILLTOADDRESS) = lBillToAddressID
        .Fields(ARINVOICES_fld_SHIPTOADDRESS) = lShipToAddressID
        .Save
        .CloseDown
    End With

    Set oARInvoice = Nothing

End Sub
```

Adding an Invoice Line Item

With the following code sample, you can create a new invoice line item.

```
Private Sub Add_ARInvoiceLineItem(ByVal lInvoiceID As Long, ByVal dtTranDate As Date, _
ByVal lPostStatus As EST_PostStatus, ByVal dtPostDate As Date, _
ByVal lCategory As EST_BillingItemTypes, ByVal lItemID As Long)

'    lInvoiceID = 7
'    dtTranDate = "09/24/2004"
'    lPostStatus = staticentryPostStatusNotPosted
'    dtPostDate = "09/30/2004"
'    lCategory = staticentry_BillingItemType_FlatRate
'    lItemID = 5 (Item Name: ADT)

    Dim oARInvoice As CARInvoice
    Set oARInvoice = New CARInvoice

    With oARInvoice
        .Init goSessionContext

        'Load the invoice for which LineItems are being added
        .Load lInvoiceID

        With .LineItems.Add
            .Fields(ARLINEITEMS_fld_TRANDATE) = dtTranDate
            .StaticEntryField(ARLINEITEMS_fld_POSTSTATUS) = lPostStatus
            .Fields(ARLINEITEMS_fld_POSTDATE) = dtPostDate
            .StaticEntryField(ARLINEITEMS_fld_CATEGORY) = lCategory
            .Fields(ARLINEITEMS_fld_AR7BILLINGITEMSID) = lItemID

            'Load the default amount and distribution specified on the billing item
            'as given below or alternatively use distribution child object
            .LoadDefaultsFromBillingItem
        End With

        .Save
        .CloseDown
    End With

    Set oARInvoice = Nothing

End Sub
```

Adding a Charge Record

With the following code sample, you can create a new charge record.

```
Private Sub Add_ARCharge(ByVal lClientID As Long, ByVal dtTranDate As Date, _
ByVal dtDueDate As Date, ByVal lPostStatus As EST_PostStatus, ByVal dtPostDate As Date, _
ByVal lCategory As EST_BillingItemTypes, ByVal lItemID As Long, ByVal sUnit As String, _
ByVal dQuantity As Double)

'    lClientID = 1
'    dtTranDate = "09/24/2004"
'    dtDueDate = "09/30/2004"
'    lPostStatus = staticentryPostStatusNotPosted
'    dtPostDate = "09/30/2004"
'    lCategory = staticentry_BillingItemType_Product
'    lItemID = 1
'    sUnit = "Box"
'    dQuantity = 5

    Dim oARCharge As CARCharge
    Set oARCharge = New CARCharge

    With oARCharge
        .Init goSessionContext
        .Fields(ARCHARGES_fld_AR7CLIENTSID) = lClientID
        .Fields(ARCHARGES_fld_TRANDATE) = dtTranDate
        .Fields(ARCHARGES_fld_DUEDATE) = dtDueDate
        .StaticEntryField(ARCHARGES_fld_POSTSTATUS) = lPostStatus
        .Fields(ARCHARGES_fld_POSTDATE) = dtPostDate
        .StaticEntryField(ARCHARGES_fld_CATEGORY) = lCategory
        .Fields(ARCHARGES_fld_AR7BILLINGITEMSID) = lItemID
        .LoadDefaultsFromBillingItem (True)
        .Fields(ARCHARGES_fld_UNITOFMEASURE) = sUnit
        .Fields(ARCHARGES_fld_QUANTITY) = dQuantity
        .RecalculateExtendedAmount
        .Save
        .CloseDown
    End With

    Set oARCharge = Nothing

End Sub
```

Adding a Credit Record

With the following code sample, you can create a new credit record.

```
Private Sub Add_ARCredit(ByVal lClientID As Long, ByVal dtTranDate As Date, _
ByVal lPostStatus As EST_PostStatus, ByVal dtPostDate As Date, _
ByVal lCategory As EST_BillingItemTypes, ByVal lItemID As Long)

'    lClientID = 5
'    dtTranDate = "09/24/2004"
'    lPostStatus = staticentryPostStatusNotPosted
'    dtPostDate = "09/30/2004"
'    lCategory = staticentry_BillingItemType_FlatRate
'    lItemID = 2

    Dim oARCredit As CARCredit
    Set oARCredit = New CARCredit

    With oARCredit
        .Init goSessionContext
        .Fields(ARCREDITS_fld_AR7CLIENTSID) = lClientID
        .Fields(ARCREDITS_fld_TRANDATE) = dtTranDate
        .StaticEntryField(ARCREDITS_fld_POSTSTATUS) = lPostStatus
        .Fields(ARCREDITS_fld_POSTDATE) = dtPostDate
        .StaticEntryField(ARCREDITS_fld_CATEGORY) = lCategory
        .Fields(ARCREDITS_fld_AR7BILLINGITEMSID) = lItemID
        .LoadDefaultsFromBillingItem (True)

        ' Define whose charges the credit will apply to.
        ' The method below will default the payee to the client
        ' the charge is created for
        .PayeeDistribution.LoadDefaultPayees
        .Save
        .CloseDown
    End With

    Set oARCredit = Nothing

End Sub
```

Adding a Billing Item Record

With the following code sample, you can create a new billing item.

```
Private Sub Add_ARBillingItem(ByVal lItemType As EST_BillingItemTypes, _
ByVal sItemID As String, ByVal lStatus As EST_ActiveStatus, ByVal sDescription As String, _
ByVal lDateDesc As EST_DateDescription, ByVal cAmount As Currency, _
ByVal bAllowUserEdit As Boolean, ByVal bTaxable As Boolean, ByVal lSalesTaxID As Long, _
bAllowTermsDiscount As Boolean, ByVal bAssessFinanceCharge As Boolean, _
ByVal sProvider As String, ByVal sComment As String)

'    lItemType = staticentry_BillingItemType_FlatRate
'    sItemID = "Flat 1"
'    lStatus = staticentryActiveStatus_Active
'    sDescription = "Description"
'    lDateDesc = staticentry_DateDescription_AnyDate
'    cAmount = 100
'    bAllowUserEdit = True
'    bTaxable = True
'    lSalesTaxID = 25
'    bAllowTermsDiscount = True
'    bAssessFinanceCharge = True
'    sProvider = "Lois"
'    sComment = "Comment"

    Dim oBillingItem As CBillingItem
    Set oBillingItem = New CBillingItem

    With oBillingItem
        .Init goSessionContext
        .StaticEntryField(BILLINGITEMS_fld_ITEMTYPE) = lItemType
        .Fields(BILLINGITEMS_fld_ITEMID) = sItemID
        .StaticEntryField(BILLINGITEMS_fld_ACTIVESTATUS) = lStatus
        .Fields(BILLINGITEMS_fld_DESCRIPTION) = sDescription
        .StaticEntryField(BILLINGITEMS_fld_DATEDESCRIPTION) = lDateDesc
        .Fields(BILLINGITEMS_fld_EXTENDEDAMOUNT) = cAmount
        .Fields(BILLINGITEMS_fld_ALLOWUSEREDIT) = bAllowUserEdit
        .Fields(BILLINGITEMS_fld_TAXABLE) = bTaxable
        .Fields(BILLINGITEMS_fld_DEFAULTSALESTAXITEMHEADERID) = lSalesTaxID
        .Fields(BILLINGITEMS_fld_ALLOWTERMSDISCOUNT) = bAllowTermsDiscount
        .Fields(BILLINGITEMS_fld_ASSESSFINANCECHARGES) = bAssessFinanceCharge
        .Fields(BILLINGITEMS_fld_SERVICEPROVIDER) = sProvider
        .Fields(BILLINGITEMS_fld_DEFAULTCOMMENT) = sComment
        .Save
        .CloseDown
    End With

    Set oBillingItem = Nothing

End Sub
```

Adding a Refund Record

With the following code sample, you can create a new refund.

```
Private Sub Add_ARRefund(ByVal lClientID As Long, ByVal dtTranDate As Date, _
ByVal lPostStatus As EST_PostStatus, ByVal dtPostDate As Date, ByVal lItemID As Long, _
ByVal cAmount As Currency, ByVal sComment As String)

'    lClientID = 5
'    dtTranDate = "09/28/2004"
'    lPostStatus = staticentryPostStatusNotPosted
'    dtPostDate = "09/30/2004"
'    lItemID = 8
'    cAmount = 40
'    sComment = "Refund Comment"

    Dim oARRefund As CARRefund
    Dim oWarningRule As IBBWarningRule

    Set oARRefund = New CARRefund

    With oARRefund
        .Init goSessionContext
        .Fields(ARREFUNDS_fld_AR7CLIENTSID) = lClientID
        .Fields(ARREFUNDS_fld_TRANDATE) = dtTranDate
        .StaticEntryField(ARREFUNDS_fld_POSTSTATUS) = lPostStatus
        .Fields(ARREFUNDS_fld_POSTDATE) = dtPostDate
        .Fields(ARREFUNDS_fld_AR7BILLINGITEMSID) = lItemID
        .LoadDefaultsFromRefundBillingItem (True)
        .Fields(ARREFUNDS_fld_AMOUNT) = cAmount
        .Fields(ARREFUNDS_fld_COMMENT) = sComment

        'Warning to apply the refund can be overridden as this can be done later
        Set oWarningRule = oARRefund
        oWarningRule.OverrideWarning_
                (ARRefund_Warning_ApplicationsDoNotExistWithBalance) = True

        .Save
        .CloseDown
    End With

    Set oARRefund = Nothing
    Set oWarningRule = Nothing

End Sub
```

Adding an Accounts Receivable Deposit

With the following code sample, you can create a new deposit.

```
Private Sub Add_ARDeposit(ByVal lSystem As EST_SystemNames, ByVal lBankID As Long, _
ByVal dtDepositDate As Date, ByVal dtEntryDate As Date, ByVal sUserDefinedID As String, _
ByVal lStatus As EST_DepositStatus, ByVal lPostStatus As EST_PostStatus, _
ByVal dtPostDate As Date, ByVal sDescription As String)

'    lSystem = bbBlackbaud_AR_System
'    lBankID = 3
'    dtDepositDate = "09/28/2004"
'    dtEntryDate = "09/28/2004"
'    sUserDefinedID = "ARD"
'    lStatus = staticentry_DepositStatus_Open
'    lPostStatus = staticentryPostStatusNotPosted
'    dtPostDate = "09/30/2004"
'    sDescription = "Test Deposit"

    Dim oDeposit As CDeposit

    Set oDeposit = New CDeposit
    With oDeposit
        .Init goSessionContext
        .StaticEntryField(DEPOSITS_fld_SYSTEMOFORIGIN) = lSystem
        .Fields(DEPOSITS_fld_BANKSID) = lBankID
        .Fields(DEPOSITS_fld_DEPOSITDATE) = dtDepositDate
        .Fields(DEPOSITS_fld_USERDEFINEDNUMBER) = sUserDefinedID
        .StaticEntryField(DEPOSITS_fld_DEPOSITSTATUS) = lStatus
        .StaticEntryField(DEPOSITS_fld_POSTSTATUS) = lPostStatus
        .Fields(DEPOSITS_fld_POSTDATE) = dtPostDate
        .Fields(DEPOSITS_fld_DESCRIPTION) = sDescription
        .Save
        .CloseDown
    End With

    Set oDeposit = Nothing

End Sub
```

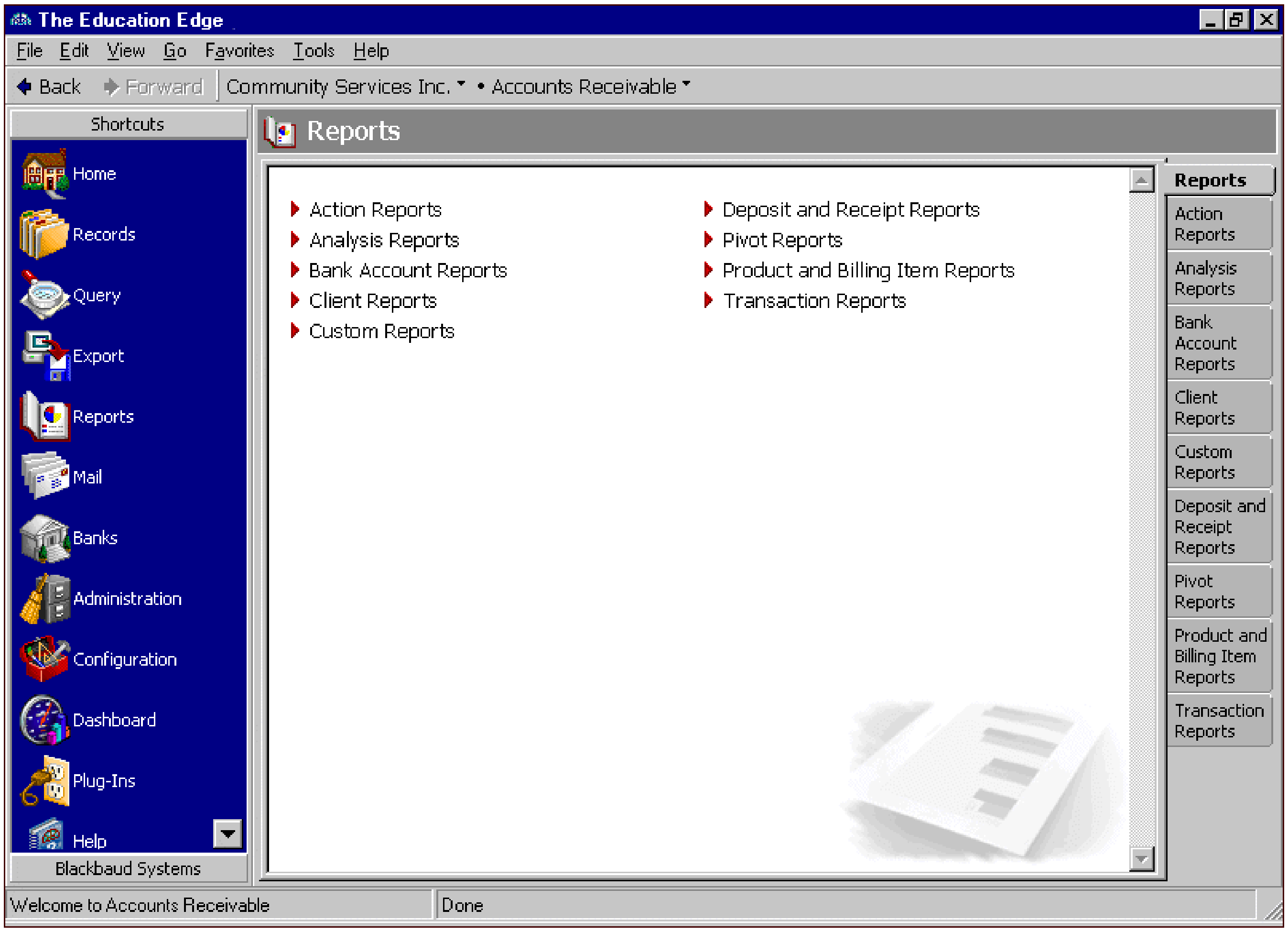
Adding an Accounts Receivable Payment

With the following code sample, you can create a new payment.

```
Private Sub Add_ARPayment(ByVal lDepositID As Long, ByVal lPayerID As Long, _
ByVal sSource As String, ByVal sComment As String, _
ByVal lReceiptStatus As EST_ReceiptStatus, ByVal lReceiptAddressID As Long, _
ByVal cAmount As Currency, ByVal cAmountBills As Currency, ByVal cAmountCoins As Currency)

'    lDepositID = 13
'    lPayerID = 11
'    sSource = "Mail"
'    sComment = "Comment"
'    lReceiptStatus = staticentry_ReceiptStatus_NotYetPrinted
'    lReceiptAddressID = 11
'    cAmount = 50
'    cAmountBills = 50
'    cAmountCoins = 0

    Dim oPaymentHeader As CPaymentHeader

    Set oPaymentHeader = New CPaymentHeader
    With oPaymentHeader
        .Init goSessionContext

        .Fields(PAYMENTHEADERS_fld_CRDEPOSITSID) = lDepositID
        .Fields(PAYMENTHEADERS_fld_PAYERID) = lPayerID
        .Fields(PAYMENTHEADERS_fld_SOURCE) = sSource
        .Fields(PAYMENTHEADERS_fld_COMMENT) = sComment
        .StaticEntryField(PAYMENTHEADERS_fld_RECEIPTSTATUS) = lReceiptStatus
        .Fields(PAYMENTHEADERS_fld_RECEIPTADDRESSID) = lReceiptAddressID

        With .PaymentComponents.Add
            .StaticEntryField(PAYMENTS_fld_PAYMENTTYPE) = staticentry_PaymentType_ARPayment
            .ApplyClientDefaults
            .Fields(PAYMENTS_fld_AMOUNT) = cAmount
            .PayeeDistribution.LoadDefaultPayees
        End With

        .Fields(PAYMENTHEADERS_fld_AMOUNT) = cAmount
        .StaticEntryField(PAYMENTHEADERS_fld_PAYMENTMETHOD) = _
                            staticentry_PaymentMethods_Cash
        .Fields(PAYMENTHEADERS_fld_AMOUNTBILLS) = cAmountBills
        .Fields(PAYMENTHEADERS_fld_AMOUNTCOINS) = cAmountCoins

        .Save
        .CloseDown
    End With

    Set oPaymentHeader = Nothing

End Sub
```

Accounts Receivable Reports Samples

This section contains code samples for creating applications you can use with ***Accounts Receivable*** reports. Samples include creating aged accounts receivable, open item, deposit list, and invoice reports.



Creating an Aged Receivable Report

With this code sample, you can create an aged receivable report.

```
Public Sub Create_AR_AgedReceivable_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Service.CreateReportInstance(bbrep_AR_AgedAccountsReceivable)

    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Aged Receivable Report Created From _
          API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_BaseAgingOn_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_TODAY

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_TranDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_LASTYEAR
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_DueDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_LASTYEAR
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_PostDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_LASTYEAR

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_IncludeBalance).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_BalanceAmount).Fields(REPORTPARAMETERVALUES_fld_CURRENCY) = 0
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_BalancePeriod).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = -1

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_AgedAR_ReduceBalance).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1
```

(Continued- page 2 of 2)

```
        'Set the various Filters
        'Set the Invoice Filter
        With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues,_
             valuenumber:=1, ValueSet:=CStr(bbFilterType_AR_Invoice))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = 1
            .Fields(FILTERS_fld_TOID) = 10
        End With
    End With

    With oReport
        .Save
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Creating an Open Item Report

With this code sample, you can create an open item report.

```
Public Sub Create_AR_OpenItem_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Service.CreateReportInstance(bbrep_AR_OpenItemReport)

    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Open Item Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_OpenItem_ChargesOpenAsOf_DateType).Fields _
          (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_TODAY
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_OpenItem_BaseOpenDateOn).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 2

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_OpenItem_TransactionDate_DateType).Fields _
          (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_OpenItem_PostDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_OpenItem_IncludeUnapplied).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) _
          = False
    End With

    With oReport
        .Save
        .CloseDown
    End With

    Set oReport = Nothing
```

(Continued- page 2 of 2)

```
    Set oMetaData = Nothing

End Sub .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
    Set oMetaData = Nothing

End Sub .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Creating a Deposit List Report

With the following code sample, you can create a deposit list report.

```
Public Sub Create_AR_DepositList_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Service.CreateReportInstance(bbrep_AR_DepositTicket)

    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Deposit List Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_DepositDateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_ALLDATES
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_EnteredOnDateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER)_
          = bbDATE_ALLDATES
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_PostDateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
          = bbDATE_ALLDATES


        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_DepositNumberFrom).Fields(REPORTPARAMETERVALUES_fld_NUMBER)_
          = 2
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
         bbrep_DepositReports_DepositNumberTo).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 5

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_IncludeAmounts).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) _
          = True
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_IncludeAmountsValue).Fields(REPORTPARAMETERVALUES_fld_
          NUMBER) = 100

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_IncludeAR).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
```

(Continued- page 2 of 2)

```
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_DepositReports_IncludeCR).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
    End With

    With oReport
        .Save
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

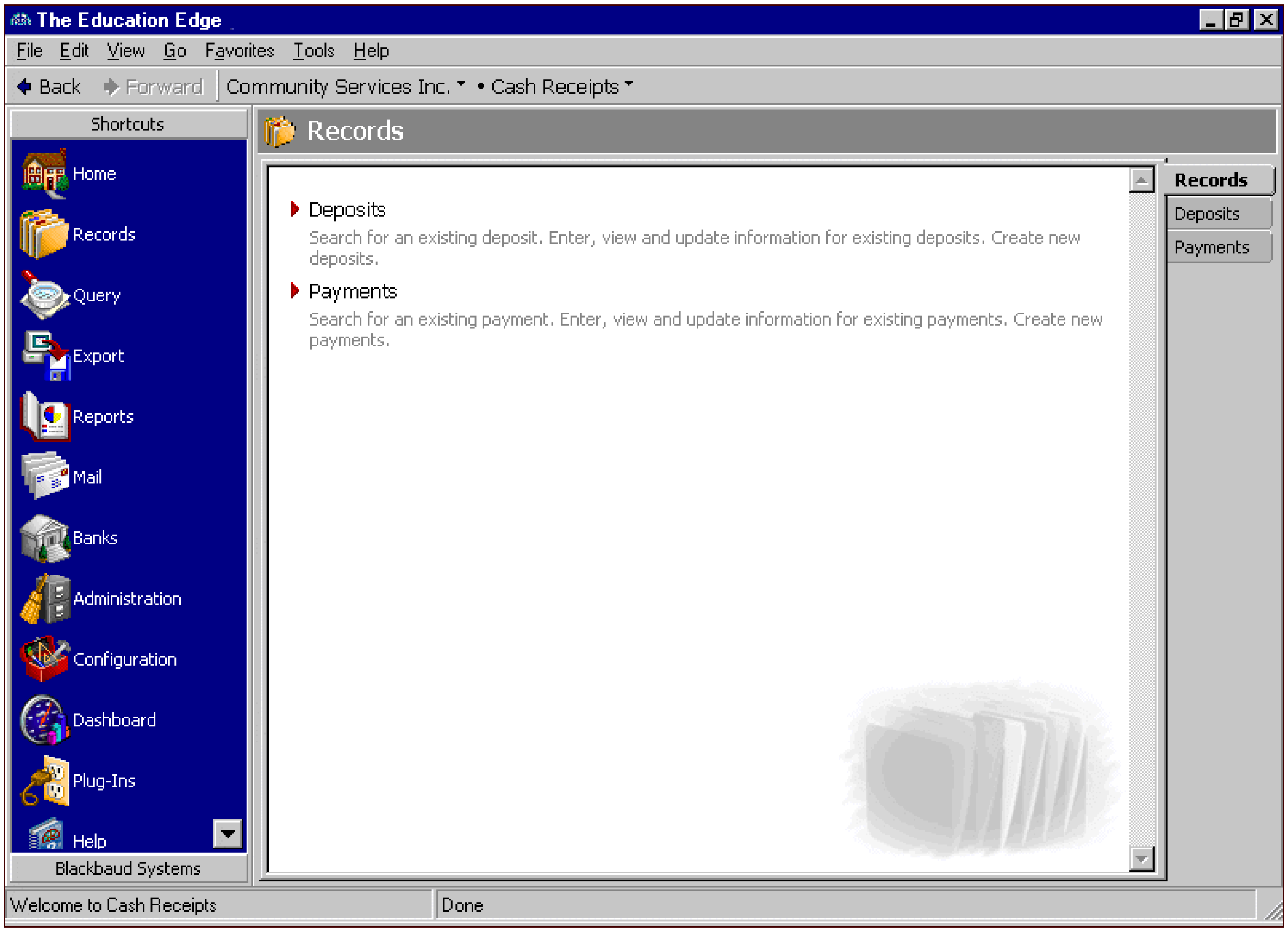
Creating an Accounts Receivable Invoice Report

With the following code sample, you can create an invoice report.

```
Public Sub Create_AR_Invoice_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Service.CreateReportInstance(bbrep_AR_InvoiceReport)

    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Invoice Report Created From API"
        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"
        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True
        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_InvoiceReport_IncludeInvoicesReturns).Fields_
            (REPORTPARAMETERVALUES_fld_NUMBER) = 1
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_InvoiceReport_DateType_InvoiceDate).Fields_
            (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_InvoiceReport_DateType_OrderedDate).Fields_
            (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_InvoiceReport_FromID).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbrep_InvoiceReport_ToID).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 10
    End With

    With oReport
        .Save
        .CloseDown
    End With

    Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Cash Receipts Records Samples

This section contains code samples for creating applications you can use with ***Cash Receipts*** records. Samples include adding deposits and payments.



Deposits
Search for an existing deposit. Enter, view and update information for existing deposits. Create new deposits.

Payments
Search for an existing payment. Enter, view and update information for existing payments. Create new payments.

Adding a Cash Receipts Deposit

With the following code sample, you can create a new deposit.

```
Private Sub Add_CRDeposit(ByVal lSystem As EST_SystemNames, ByVal lBankID As Long, _
ByVal dtDepositDate As Date, ByVal dtEntryDate As Date, ByVal sUserDefinedID As String, _
ByVal lStatus As EST_DepositStatus, ByVal lPostStatus As EST_PostStatus, _
ByVal dtPostDate As Date, ByVal sDescription As String)

'    lSystem = bbBlackbaud_CR_System
'    lBankID = 3
'    dtDepositDate = "09/28/2004"
'    dtEntryDate = "09/28/2004"
'    sUserDefinedID = "CR"
'    lStatus = staticentry_DepositStatus_Open
'    lPostStatus = staticentryPostStatusNotPosted
'    dtPostDate = "09/30/2004"
'    sDescription = "CR Test Deposit"

    Dim oDeposit As CDeposit

    Set oDeposit = New CDeposit
    With oDeposit
        .Init goSessionContext
        .StaticEntryField(DEPOSITS_fld_SYSTEMOFORIGIN) = lSystem
        .Fields(DEPOSITS_fld_BANKSID) = lBankID
        .Fields(DEPOSITS_fld_DEPOSITDATE) = dtDepositDate
        .Fields(DEPOSITS_fld_DATEENTERED) = dtEntryDate
        .Fields(DEPOSITS_fld_USERDEFINEDNUMBER) = sUserDefinedID
        .StaticEntryField(DEPOSITS_fld_DEPOSITSTATUS) = lStatus
        .StaticEntryField(DEPOSITS_fld_POSTSTATUS) = lPostStatus
        .Fields(DEPOSITS_fld_POSTDATE) = dtPostDate
        .Fields(DEPOSITS_fld_DESCRIPTION) = sDescription
        .Save
        .CloseDown
    End With

    Set oDeposit = Nothing

End Sub
```

Adding a Cash Receipts Payment

With the following code sample, you can create a new payment.

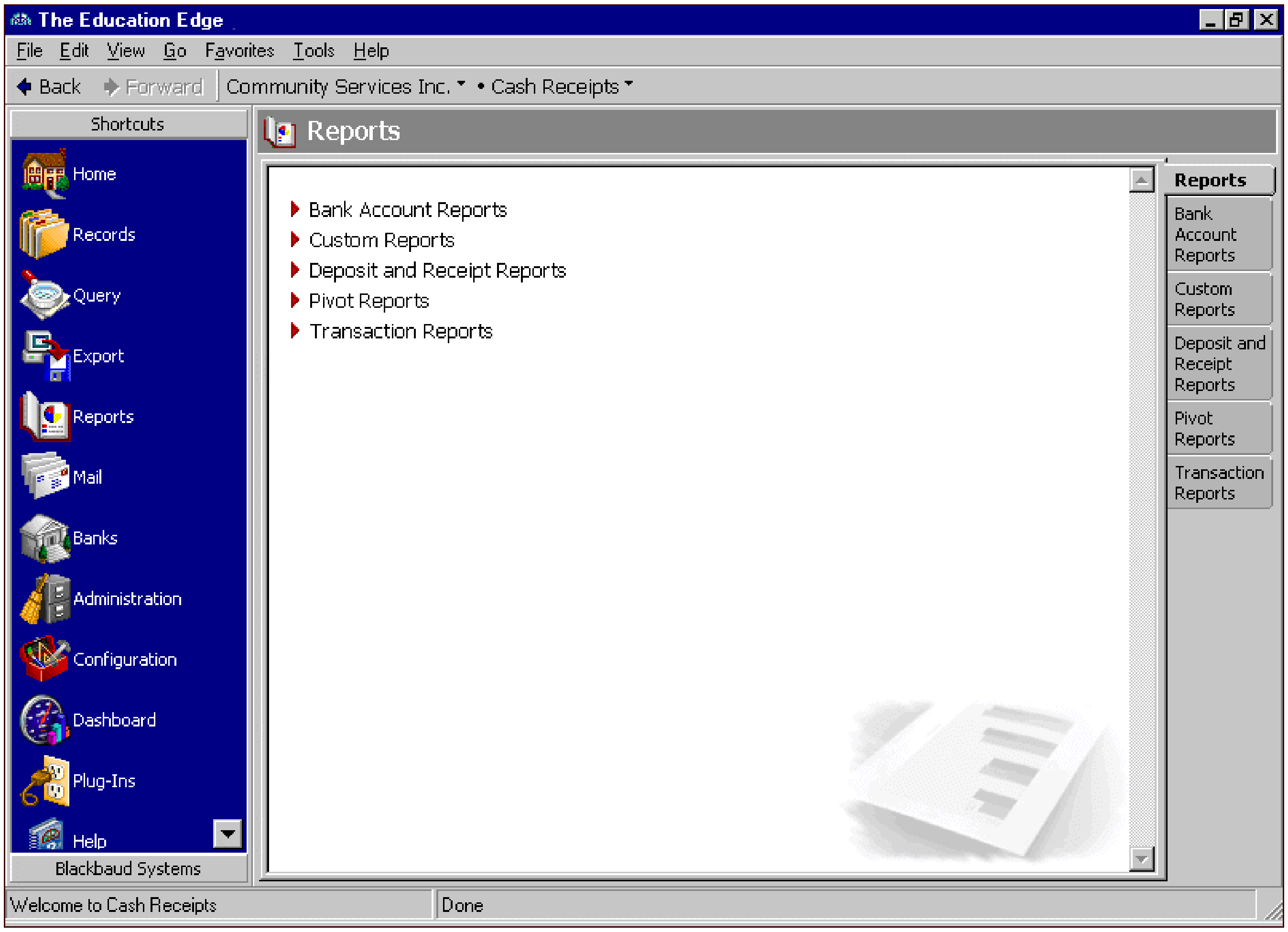
```
Private Sub Add_CRPayment(ByVal lDepositID As Long, ByVal sPayerName As String,  _
 ByVal dtTranDate As Date, ByVal cAmount As Currency, ByVal sSource As String,  _
 ByVal sDescription As String, ByVal lReceiptStatus As EST_ReceiptStatus,  _
 ByVal sCreditAcctNum As String, ByVal sProject As String, ByVal sClass As String,  _
 ByVal sTranCode1 As String, ByVal sTranCode2 As String, ByVal cAmountBills As Currency, _
 ByVal cAmountCoins As Currency)

'    lDepositID = 19
'    sPayerName = "CR Payer"
'    dtTranDate = "09/29/2004"
'    cAmount = 60
'    sSource = "Test"
'    sDescription = "CR Payment"
'    lReceiptStatus = staticentry_ReceiptStatus_NotYetPrinted
'    sCreditAcctNum = "01-4050-04"
'    sClass = "Unrestricted Net Assets"
'    sProject = "1001"
'    sTranCode1 = "None"
'    sTranCode2 = "Spendable"
'    cAmountBills = 60
'    cAmountCoins = 0

    Dim oPaymentHeader As CPaymentHeader

    Set oPaymentHeader = New CPaymentHeader
    With oPaymentHeader
        .Init goSessionContext
        .Fields(PAYMENTHEADERS_fld_CRDEPOSITSID) = lDepositID
        .Fields(PAYMENTHEADERS_fld_TRANDATE) = dtTranDate
        .Fields(PAYMENTHEADERS_fld_AMOUNT) = cAmount
        .Fields(PAYMENTHEADERS_fld_DESCRIPTION) = sDescription
        .StaticEntryField(PAYMENTHEADERS_fld_RECEIPTSTATUS) = lReceiptStatus

        With .PaymentComponents.Add
            .StaticEntryField(PAYMENTS_fld_PAYMENTTYPE) = staticentry_PaymentType_CRPayment
            .Fields(PAYMENTS_fld_PAYERNAME) = sPayerName
            .Fields(PAYMENTS_fld_SOURCE) = sSource
            .Fields(PAYMENTS_fld_AMOUNT) = cAmount
        End With

        With .BaseComponent.Distribution.Add
            .Fields(BBDISTRIBUTIONS_fld_CREDITACCTNUM) = sCreditAcctNum
            .Fields(BBDISTRIBUTIONS_fld_AMOUNT) = cAmount
            With .TransactionDistributions(bbTranDistType_Credit).Add
                .Fields(BBTRANSACTIONDISTRIBUTIONS_fld_GL7PROJECTSID) = sProject
                .Fields(BBTRANSACTIONDISTRIBUTIONS_fld_CLASS) = sClass
                .Fields(BBTRANSACTIONDISTRIBUTIONS_fld_TRANSACTIONCODE1) = sTranCode1
                .Fields(BBTRANSACTIONDISTRIBUTIONS_fld_TRANSACTIONCODE2) = sTranCode2
                .Fields(BBTRANSACTIONDISTRIBUTIONS_fld_AMOUNT) = cAmount
            End With
```

(Continued- page 2 of 2)

```
         End With

         .StaticEntryField(PAYMENTHEADERS_fld_PAYMENTMETHOD) _
           = staticentry_PaymentMethods_Cash
         .Fields(PAYMENTHEADERS_fld_AMOUNTBILLS) = cAmountBills
         .Fields(PAYMENTHEADERS_fld_AMOUNTCOINS) = cAmountCoins

         .Save
         .CloseDown
     End With

     Set oPaymentHeader = Nothing

End Sub
```

Cash Receipts Reports Samples

This section contains code samples for creating applications you can use with ***Cash Receipts*** reports. There is a sample for creating a cash receipts report.



Creating a Cash Receipts Report

With the following code sample, you can create a cash receipts report.

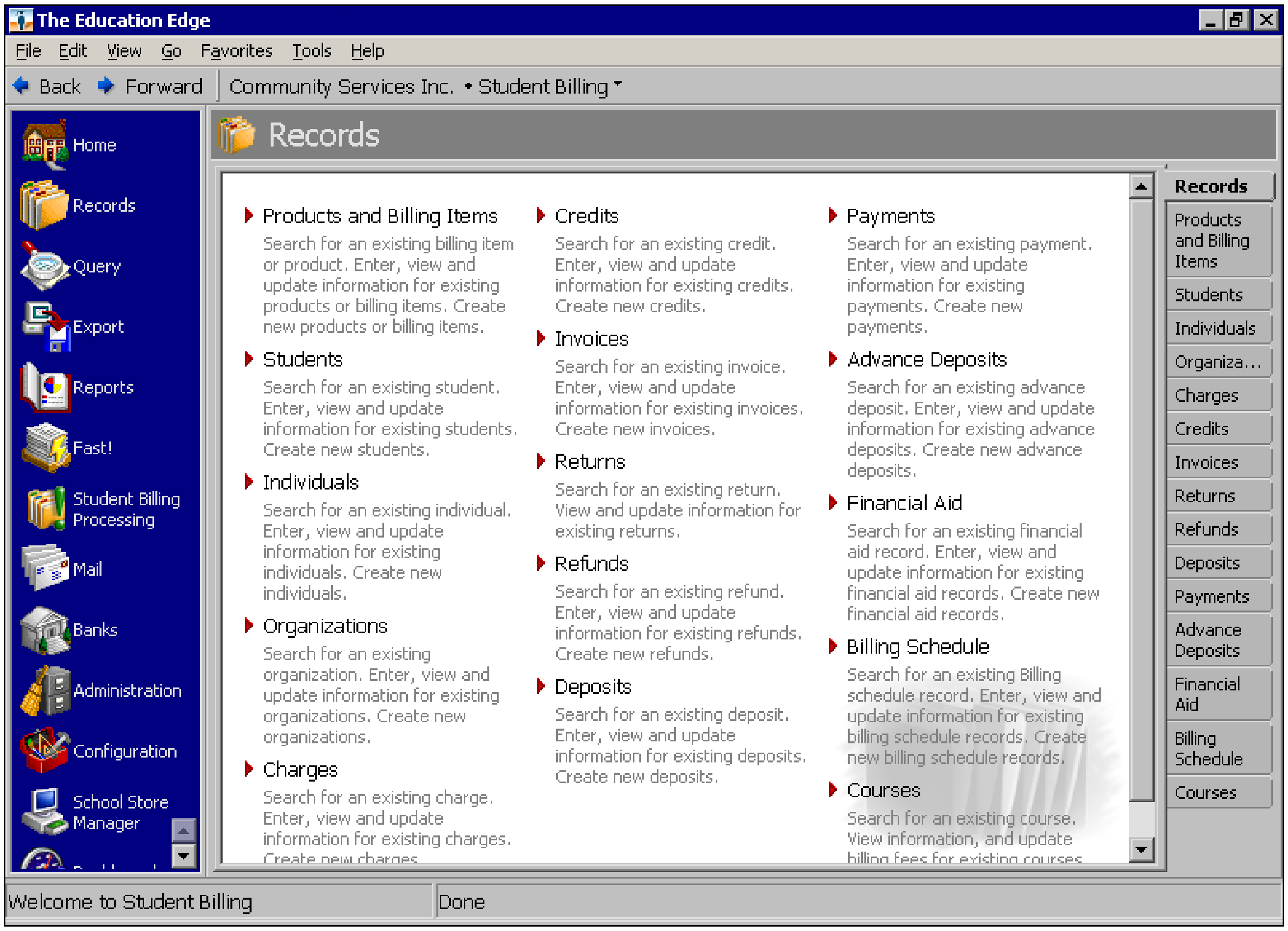
```
Public Sub Create_CR_Receipts_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Service.CreateReportInstance(bbrep_CR_Receipts)

    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
         .Property(REPORTPARAMETERNAMES_fld_NAME) = "Cash Receipts Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With

    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_DepositDate_DateType).Fields_
           (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_ReceiptDate_DateType).Fields_
           (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_EnteredOnDate_DateType).Fields_
           (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
         bbRep_CashReceipts_DepositNumberFrom).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_DepositNumberTo).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 3

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
         bbRep_CashReceipts_ReceiptNumberFrom).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_ReceiptNumberTo).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 10

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_ShowPaymentDetails).Fields_
           (REPORTPARAMETERVALUES_fld_BOOLEAN) = True
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_IncludeAR).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
```

(Continued- page 2 of 2)

```
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
          bbRep_CashReceipts_IncludeCR).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
    End With

    With oReport
        .Save
        .CloseDown
    End With

Set oReport = Nothing
    Set oMetaData = Nothing

End Sub
```

Student Billing Records Samples

This section contains code samples for creating applications you can use with ***Student Billing*** records. Samples include adding products and billing items, clients, charges, credits, invoices, recurring invoices, returns, refunds, deposits, and payments.



Products and Billing Items
Search for an existing billing item or product. Enter, view and update information for existing products or billing items. Create new products or billing items.

Students
Search for an existing student. Enter, view and update information for existing students. Create new students.

Individuals
Search for an existing individual. Enter, view and update information for existing individuals. Create new individuals.

Organizations
Search for an existing organization. Enter, view and update information for existing organizations. Create new organizations.

Charges
Search for an existing charge. Enter, view and update information for existing charges. Create new charges.

Credits
Search for an existing credit. Enter, view and update information for existing credits. Create new credits.

Invoices
Search for an existing invoice. Enter, view and update information for existing invoices. Create new invoices.

Returns
Search for an existing return. View and update information for existing returns.

Refunds
Search for an existing refund. Enter, view and update information for existing refunds. Create new refunds.

Deposits
Search for an existing deposit. Enter, view and update information for existing deposits. Create new deposits.

Payments
Search for an existing payment. Enter, view and update information for existing payments. Create new payments.

Advance Deposits
Search for an existing advance deposit. Enter, view and update information for existing advance deposits. Create new advance deposits.

Financial Aid
Search for an existing financial aid record. Enter, view and update information for existing financial aid records. Create new financial aid records.

Billing Schedule
Search for an existing Billing schedule record. Enter, view and update information for existing billing schedule records. Create new billing schedule records.

Courses
Search for an existing course. View information, and update billing fees for existing courses.

Adding an Advance Deposit Record

With the following code sample, you can add an advance deposit record.

```
Private Sub Add_SBAdvanceDeposit(ByVal lEA7RecordsID As Long, ByVal dtTranDate As Date, _
ByVal dtDueDate As Date, ByVal lItemID As Long, cAmount As Currency, _
ByVal sComment As String)

' lEA7RecordsID = 5
' dtTranDate = "09/24/2006"
' dtDueDate = "09/30/2006"
' lItemID = 60
' cAmount = 25
' sComment = "Advance Deposit"

    Dim oSBAdvanceDeposit As CSBAdvanceDeposit
    Set oSBAdvanceDeposit = New CSBAdvanceDeposit

    With oSBAdvanceDeposit
        .Init goSessionContext
        .Fields(SBADVANCEDEPOSIT_fld_EA7RECORDSID) = lEA7RecordsID
        .Fields(SBADVANCEDEPOSIT_fld_TRANDATE) = dtTranDate
        .Fields(SBADVANCEDEPOSIT_fld_DUEDATE) = dtDueDate
        .Fields(SBADVANCEDEPOSIT_fld_SBBILLINGITEMSID) = lItemID
        .LoadDefaultsFromBillingItem (True)
        .Fields(SBADVANCEDEPOSIT_fld_AMOUNT) = cAmount
        .Fields(SBADVANCEDEPOSIT_fld_COMMENT) = sComment

        .ApplyRecordDefaults

        .Save
        .CloseDown
    End With
    Set oSBAdvanceDeposit = Nothing

End Sub
```

Adding a Billing Item

The following code sample illustrates how to add a flat rate billing item in ***Student Billing***. You can also use this sample to add other billing items by changing the object and its fields and required field entries.

```
Private Sub Add_SBBillingItem(ByVal lItemType As EST_BillingItemTypes, ByVal sItemID As _
String, ByVal lStatus As EST_ActiveStatus, ByVal sDescription As String, ByVal lDateDesc _
As EST_DateDescription, ByVal cAmount As Currency, ByVal bAllowUserEdit As Boolean, _
ByVal bTaxable As Boolean, ByVal lSalesTaxID As Long, bAllowTermsDiscount As Boolean, _
ByVal bAssessFinanceCharge As Boolean, ByVal sComment As String)

' lItemType = staticentry_BillingItemType_FlatRate
' sItemID = "Flat 1"
' lStatus = staticentryActiveStatus_Active
' sDescription = "Description"
' lDateDesc = staticentry_DateDescription_AnyDate
' cAmount = 100
' bAllowUserEdit = True
' bTaxable = True
' lSalesTaxID = 3
' bAllowTermsDiscount = True
' bAssessFinanceCharge = True
' sComment = "Comment"

    Dim oBillingItem As CBillingItem
    Set oBillingItem = New CBillingItem

    With oBillingItem
        .Init goSessionContext, bbBlackbaud_SB_System

        .StaticEntryField(BILLINGITEMS_fld_ITEMTYPE) = lItemType
        .Fields(BILLINGITEMS_fld_ITEMID) = sItemID
        .StaticEntryField(BILLINGITEMS_fld_ACTIVESTATUS) = lStatus
        .Fields(BILLINGITEMS_fld_DESCRIPTION) = sDescription
        .StaticEntryField(BILLINGITEMS_fld_DATEDESCRIPTION) = lDateDesc
        .Fields(BILLINGITEMS_fld_EXTENDEDAMOUNT) = cAmount
        .Fields(BILLINGITEMS_fld_ALLOWUSEREDIT) = bAllowUserEdit
        .Fields(BILLINGITEMS_fld_TAXABLE) = bTaxable
        .Fields(BILLINGITEMS_fld_DEFAULTSALESTAXITEMHEADERID) = lSalesTaxID
        .Fields(BILLINGITEMS_fld_ALLOWTERMSDISCOUNT) = bAllowTermsDiscount
        .Fields(BILLINGITEMS_fld_ASSESSFINANCECHARGES) = bAssessFinanceCharge
        .Fields(BILLINGITEMS_fld_DEFAULTCOMMENT) = sComment
        .Save
        .CloseDown
    End With

    Set oBillingItem = Nothing
End Sub
```

Adding a Billing Schedule Record

With the following code sample, you can add a billing schedule record in ***Student Billing***.

```
Private Sub Add_SBBillingSchedule(ByVal lEA7RecordsID As Long, ByVal dtTranDate As Date, _
ByVal lCategory As EST_BillingItemTypes, ByVal lItemID As Long, cAmount As Currency, _
ByVal sComment As String)

' lEA7RecordsID = 5
' dtTranDate = "09/24/2006"
' lCategory = staticentry_BillingItemType_AutomaticFees_SingleAmountAndSchedule
' lItemID = 12
' cAmount = 25
' sComment = "Billing Schedule"

    Dim oSBBillingSchedule As CSBBillingschedule
    Set oSBBillingSchedule = New CSBBillingschedule

    With oSBBillingSchedule
        .Init goSessionContext
        .Fields(SBBILLINGSCHEDULES_fld_EA7RECORDSID) = lEA7RecordsID
        .Fields(SBBILLINGSCHEDULES_fld_TRANDATE) = dtTranDate
        .StaticEntryField(SBBILLINGSCHEDULES_fld_CATEGORY) = lCategory
        .Fields(SBBILLINGSCHEDULES_fld_SB7BILLINGITEMSID) = lItemID

        .LoadDefaultsFromBillingItem (True)

        .Fields(SBBILLINGSCHEDULES_fld_AMOUNT) = cAmount
        .Fields(SBBILLINGSCHEDULES_fld_COMMENT) = sComment

        .ApplyRecordDefaults

        .Save
        .CloseDown
    End With
    Set oSBBillingSchedule = Nothing

End Sub
```

Adding a Charge Record

With the following code sample, you can add charges to a student, individual, or organization record in ***Student Billing***.

```
Private Sub Add_SBCharge(ByVal lEA7RecordsID As Long, ByVal dtTranDate As Date, _
ByVal dtDueDate As Date, ByVal lPostStatus As EST_PostStatus, ByVal dtPostDate As Date, _
ByVal lCategory As EST_BillingItemTypes, ByVal lItemID As Long, ByVal sUnit As String, _
ByVal dQuantity As Double)

' lEA7RecordsID = 1
' dtTranDate = "06/24/2006"
' dtDueDate = "06/30/2006"
' lPostStatus = staticentryPostStatusNotPosted
' dtPostDate = "06/30/2003"
' lCategory = staticentry_BillingItemType_PerUsage
' lItemID = 38
' sUnit = "Box"
' dQuantity = 5

    Dim oSBCharge As CSBCharge
    Set oSBCharge = New CSBCharge

    With oSBCharge
        .Init goSessionContext
        .Fields(SBCHARGES_fld_EA7RECORDSID) = lEA7RecordsID
        .Fields(SBCHARGES_fld_TRANDATE) = dtTranDate
        .Fields(SBCHARGES_fld_DUEDATE) = dtDueDate
        .StaticEntryField(SBCHARGES_fld_POSTSTATUS) = lPostStatus
        .Fields(SBCHARGES_fld_POSTDATE) = dtPostDate
        .StaticEntryField(SBCHARGES_fld_CATEGORY) = lCategory
        .Fields(SBCHARGES_fld_SB7BILLINGITEMSID) = lItemID
        .LoadDefaultsFromBillingItem (True)
        .Fields(SBCHARGES_fld_UNITOFMEASURE) = sUnit
        .Fields(SBCHARGES_fld_QUANTITY) = dQuantity
        .RecalculateExtendedAmount
        .Save
        .CloseDown
    End With
    Set oSBCharge = Nothing

End Sub
```

Adding a Credit Record

With the following code sample, you can add credits to a student, individual, or organization record in ***Student Billing***.

```
Private Sub Add_SBCredit(ByVal lEA7RecordID As Long, ByVal dtTranDate As Date, _
ByVal lPostStatus As EST_PostStatus, ByVal dtPostDate As Date, _
ByVal lCategory As EST_BillingItemTypes, ByVal lItemID As Long)

' lEA7RecordID = 5
' dtTranDate = "09/24/2006"
' lPostStatus = staticentryPostStatusNotPosted
' dtPostDate = "09/30/2006"
' lCategory = staticentry_BillingItemType_FlatRate
' lItemID = 37

    Dim oSBCredit As CSBCredit
    Set oSBCredit = New CSBCredit

    With oSBCredit
        .Init goSessionContext
        .Fields(SBCREDITS_fld_EA7RECORDSID) = lEA7RecordID
        .Fields(SBCREDITS_fld_TRANDATE) = dtTranDate
        .StaticEntryField(SBCREDITS_fld_POSTSTATUS) = lPostStatus
        .Fields(SBCREDITS_fld_POSTDATE) = dtPostDate
        .StaticEntryField(SBCREDITS_fld_CATEGORY) = lCategory
        .Fields(SBCREDITS_fld_SB7BILLINGITEMSID) = lItemID
        .LoadDefaultsFromBillingItem (True)
        ' Define whose charges the credit will apply to.
        ' The method below will default the payee to the record
        ' the charge is created for
        .PayeeDistribution.LoadDefaultPayees
        .Save
        .CloseDown
    End With

    Set oSBCredit = Nothing

End Sub
```

Adding a Deposit Record

With the following code sample, you can add a deposit.

```
Private Sub Add_SBDeposit(ByVal lSystem As EST_SystemNames, ByVal lBankID As Long, _
ByVal dtDepositDate As Date, ByVal dtEntryDate As Date, ByVal sUserDefinedID As String, _
ByVal lStatus As EST_DepositStatus, ByVal lPostStatus As EST_PostStatus, _
ByVal dtPostDate As Date, ByVal sDescription As String)

' lSystem = bbBlackbaud_SB_System
' lBankID = 3
' dtDepositDate = "09/28/2006"
' dtEntryDate = "09/28/2006"
' sUserDefinedID = "SBD"
' lStatus = staticentry_DepositStatus_Open
' lPostStatus = staticentryPostStatusNotPosted
' dtPostDate = "09/30/2006"
' sDescription = "Test Deposit"

    Dim oDeposit As CDeposit
    Set oDeposit = New CDeposit

    With oDeposit
        .Init goSessionContext
        .StaticEntryField(DEPOSITS_fld_SYSTEMOFORIGIN) = lSystem
        .Fields(DEPOSITS_fld_BANKSID) = lBankID
        .Fields(DEPOSITS_fld_DEPOSITDATE) = dtDepositDate
        .Fields(DEPOSITS_fld_USERDEFINEDNUMBER) = sUserDefinedID
        .StaticEntryField(DEPOSITS_fld_DEPOSITSTATUS) = lStatus
        .StaticEntryField(DEPOSITS_fld_POSTSTATUS) = lPostStatus
        .Fields(DEPOSITS_fld_POSTDATE) = dtPostDate
        .Fields(DEPOSITS_fld_DESCRIPTION) = sDescription
        .Save
        .CloseDown
    End With

    Set oDeposit = Nothing
End Sub
```

Adding a Financial Aid Record

With the following code sample, you can add a financial aid record in ***Student Billing***.

```
Private Sub Add_SBFinancialAid(ByVal lEA7RecordsID As Long, ByVal dtTranDate As Date, _
ByVal lItemID As Long, cAmount As Currency, ByVal sComment As String)

' lEA7RecordsID = 5
' dtTranDate = "09/24/2006"
' lItemID = 57
' cAmount = 25
' sComment = "Financial Aid"

    Dim oSBFinancialAid As CSBFinancialAid
    Set oSBFinancialAid = New CSBFinancialAid

    With oSBFinancialAid
        .Init goSessionContext
        .Fields(SBFINANCIALAIDS_fld_EA7RECORDSID) = lEA7RecordsID
        .Fields(SBFINANCIALAIDS_fld_TRANDATE) = dtTranDate
        .Fields(SBFINANCIALAIDS_fld_SB7BILLINGITEMSID) = lItemID

        .LoadDefaultsFromBillingItem (True)

        .Fields(SBFINANCIALAIDS_fld_AMOUNT) = cAmount
        .Fields(SBFINANCIALAIDS_fld_COMMENT) = sComment

        .ApplyRecordDefaults

        .Save
        .CloseDown
    End With
    Set oSBFinancialAid = Nothing

End Sub
```

Adding an Individual Record

With the following code sample, you can add a ***Student Billing*** individual. This sample also demonstrates how to override the warning that appears if you create an individual record without adding a primary address.

```
Private Sub Add_SBIndividual(ByVal sFirstName As String, ByVal sLastName As String, _
               ByVal sIndividualID As String, ByVal sStatus As String,_
               ByVal sGender As String, _ByVal sSSN As String, ByVal dtBirthDate As Date, _
               ByVal sReligion As String, ByVal sEthnicity As String)

' sFirstName = "Individual First"
' sLastName = "Individual Last"
' sIndividualID = "1"
' sStatus = "Active"
' sGender = "Male"
' sSSN = "123-45-6784"
' dtBirthDate = "09/12/1973"
' sReligion = "Baptist"
' sEthnicity = "Caucasian"

    Dim oIndividual As cSBIndividualRecord
    Dim oWarningRule As IBBWarningRule

    Set oIndividual = New cSBIndividualRecord
    With oIndividual
        .Init goSessionContext

        .Fields(SBINDIVIDUALS_fld_FIRSTNAME) = sFirstName
        .Fields(SBINDIVIDUALS_fld_LASTNAME) = sLastName
        .Fields(SBINDIVIDUALS_fld_USERDEFINEDID) = sIndividualID
        .Fields(SBINDIVIDUALS_fld_BILLINGSTATUS) = sStatus
        .Fields(SBINDIVIDUALS_fld_GENDER) = sGender
        .Fields(SBINDIVIDUALS_fld_SSN) = sSSN
        .Fields(SBINDIVIDUALS_fld_BIRTHDATE) = dtBirthDate
        .Fields(SBINDIVIDUALS_fld_RELIGION) = sReligion
        .Fields(SBINDIVIDUALS_fld_ETHNICITY) = sEthnicity

        'This needs to be done if you want to override warning rule given regarding the _
        'primary address if one does not exist
        Set oWarningRule = oIndividual
        oWarningRule.OverrideWarning(SBINDIVIDUAL_Warning_PrimaryAddress) = True

        .Save
        .CloseDown
    End With

    Set oIndividual = Nothing
    Set oWarningRule = Nothing
End Sub
```

Adding an Organization Record

With the following code sample, you can add a ***Student Billing*** organization record. This sample also demonstrates how to override the warning that appears if you create an individual record without adding a primary address.

```
Private Sub Add_SBOrganization(ByVal sOrganizationName As String, _
ByVal sOrganizationID As String, ByVal sStatus As String, ByVal sClassification As String, _
ByVal sType As String, ByVal sIndustry As String, ByVal sCFDA As String)

' sOrganizationName = "Organization Name"
' sOrganizationID = "1"
' sStatus = "Active"
' sClassification = "School"
' sType = "College"
' sIndustry = "Arts"
' sCFDA = "CFDA 1"

    Dim oOrganization As CSBOrganization
    Dim oWarningRule As IBBWarningRule

    Set oOrganization = New CSBOrganization
    With oOrganization
        .Init goSessionContext

        .Fields(SBORGANIZATIONS_fld_ORGANIZATIONNAME) = sOrganizationName
        .Fields(SBORGANIZATIONS_fld_USERDEFINEDID) = sOrganizationID
        .Fields(SBORGANIZATIONS_fld_BILLINGSTATUS) = sStatus
        .Fields(SBORGANIZATIONS_fld_CLASSIFICATION) = sClassification
        .Fields(SBORGANIZATIONS_fld_TYPE) = sType
        .Fields(SBORGANIZATIONS_fld_INDUSTRY) = sIndustry
        .Fields(SBORGANIZATIONS_fld_CFDA) = sCFDA

        'This needs to be done if you want to override warning rule given regarding the _
        'primary address if one does not exist
        Set oWarningRule = oOrganization
        oWarningRule.OverrideWarning(SBOrganization_Warning_PrimaryAddress) = True
        .Save
        .CloseDown
    End With

    Set oOrganization = Nothing
    Set oWarningRule = Nothing

End Sub
```

Adding a Payment Record

With the following code sample, you can add payments in ***Student Billing***.

```
Private Sub Add_SBPayment(ByVal lDepositID As Long, ByVal lEA7RecordsID As Long, _
ByVal sSource As String, ByVal sComment As String, _
ByVal lReceiptStatus As EST_ReceiptStatus, ByVal lReceiptAddressID As Long, _
ByVal cAmount As Currency, ByVal cAmountBills As Currency, ByVal cAmountCoins As Currency)

' lDepositID = 21
' lEA7RecordsID = 5
' sSource = "Mail"
' sComment = "Comment"
' lReceiptStatus = staticentry_ReceiptStatus_NotYetPrinted
' lReceiptAddressID = 11
' cAmount = 50
' cAmountBills = 50
' cAmountCoins = 0

    Dim oPaymentHeader As CPaymentHeader
    Set oPaymentHeader = New CPaymentHeader

    With oPaymentHeader
        .Init goSessionContext
        .Fields(PAYMENTHEADERS_fld_CRDEPOSITSID) = lDepositID
        .Fields(PAYMENTHEADERS_fld_EA7RECORDSID) = lEA7RecordsID
        .Fields(PAYMENTHEADERS_fld_SOURCE) = sSource
        .Fields(PAYMENTHEADERS_fld_COMMENT) = sComment
        .StaticEntryField(PAYMENTHEADERS_fld_RECEIPTSTATUS) = lReceiptStatus
        .Fields(PAYMENTHEADERS_fld_RECEIPTADDRESSID) = lReceiptAddressID

        With .PaymentComponents.Add
            .StaticEntryField(PAYMENTS_fld_PAYMENTTYPE) = staticentry_PaymentType_SBPayment
            .ApplyClientDefaults
            .Fields(PAYMENTS_fld_AMOUNT) = cAmount
            .PayeeDistribution.LoadDefaultPayees
        End With

        .Fields(PAYMENTHEADERS_fld_AMOUNT) = cAmount
        .StaticEntryField(PAYMENTHEADERS_fld_PAYMENTMETHOD) = staticentry_
           PaymentMethods_Cash
        .Fields(PAYMENTHEADERS_fld_AMOUNTBILLS) = cAmountBills
        .Fields(PAYMENTHEADERS_fld_AMOUNTCOINS) = cAmountCoins
        .Save
        .CloseDown
    End With

    Set oPaymentHeader = Nothing
End Sub
```

Adding a Refund Record

With the following code sample, you can add a student record.

```
Private Sub Add_SBStudent(ByVal sFirstName As String, ByVal sLastName As String, _
                          ByVal sStudentID As String, ByVal sStatus As String, _
                          ByVal sGradeLevel As String, ByVal sGender As String, _
                          ByVal sSSN As String, ByVal dtBirthDate As Date, _
                          ByVal sReligion As String, ByVal sEthnicity As String)

' sFirstName = "Student First"
' sLastName = "Student Last"
' sStudentID = "1"
' sStatus = "Active"
' sGradeLevel = "PK"
' sGender = "Male"
' sSSN = "123-45-6783"
' dtBirthDate = "08/12/2001
' sReligion = "Baptist"
' sEthnicity = "Caucasian"

    Dim oStudent As cSBStudent
    Dim oWarningRule As IBBWarningRule
    Set oStudent = New cSBStudent
    With oStudent
        .Init goSessionContext

        .Fields(SBSTUDENTS_fld_FIRSTNAME) = sFirstName
        .Fields(SBSTUDENTS_fld_LASTNAME) = sLastName
        .Fields(SBSTUDENTS_fld_USERDEFINEDID) = sStudentID
        .Fields(SBSTUDENTS_fld_BILLINGSTATUS) = sStatus
        .Fields(SBSTUDENTS_fld_GRADELEVEL) = sGradeLevel
        .Fields(SBSTUDENTS_fld_GENDER) = sGender
        .Fields(SBSTUDENTS_fld_SSN) = sSSN
        .Fields(SBSTUDENTS_fld_BIRTHDATE) = dtBirthDate
        .Fields(SBSTUDENTS_fld_RELIGION) = sReligion
        .Fields(SBSTUDENTS_fld_ETHNICITY) = sEthnicity

        .StaticEntryField(SBSTUDENTS_fld_BILLINGOPTION) = _
        staticentry_SBR_SB_STU_UseThisBillingOption_AssignToStudent

        With .Statements.Add
            .Fields(STATEMENT_fld_PAYERID) = 2
            .StaticEntryField(STATEMENT_fld_STATEMENTTYPE) = _
            staticentry_StatementCharges_AllCharges
        End With

        'This needs to be done if you want to override warning rule given regarding the _
        'primary address if one does not exist
```

Adding a student record, continued (page 2 of 2)

```
Set oWarningRule = oStudent
        oWarningRule.OverrideWarning(SBStudent_Warning_PrimaryAddress) = True

        .Save
        .CloseDown
End With
    Set oStudent = Nothing
    Set oWarningRule = Nothing
End Sub
```

Adding a Student Record

With the following code sample, you can add a student record.

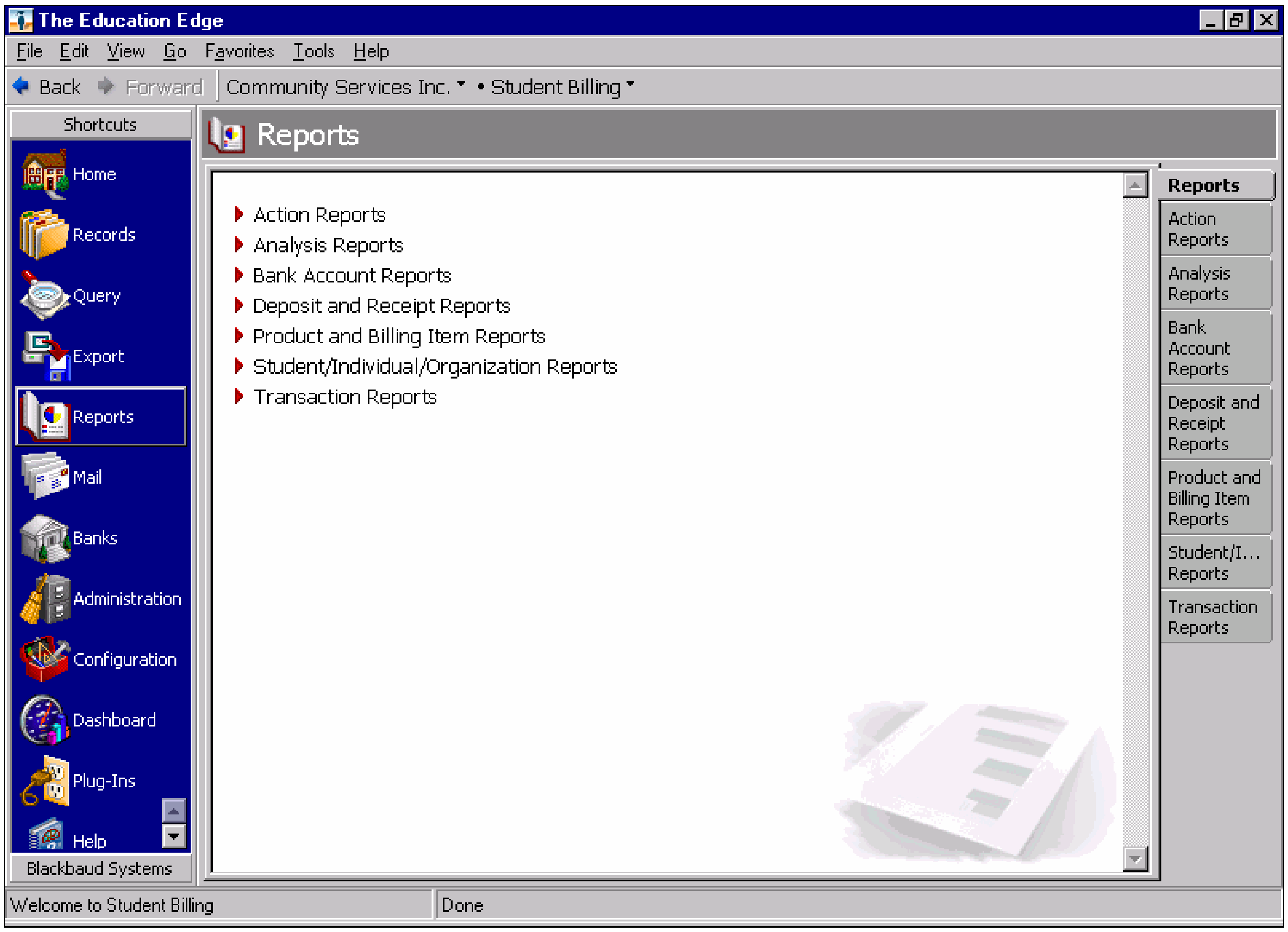
```
Private Sub Add_SBStudent(ByVal sFirstName As String, ByVal sLastName As String, _
                          ByVal sStudentID As String, ByVal sStatus As String, _
                          ByVal sGradeLevel As String, ByVal sGender As String, _
                          ByVal sSSN As String, ByVal dtBirthDate As Date, _
                          ByVal sReligion As String, ByVal sEthnicity As String)

' sFirstName = "Student First"
' sLastName = "Student Last"
' sStudentID = "1"
' sStatus = "Active"
' sGradeLevel = "PK"
' sGender = "Male"
' sSSN = "123-45-6783"
' dtBirthDate = "08/12/2001
' sReligion = "Baptist"
' sEthnicity = "Caucasian"

    Dim oStudent As cSBStudent
    Dim oWarningRule As IBBWarningRule
    Set oStudent = New cSBStudent
    With oStudent
        .Init goSessionContext

        .Fields(SBSTUDENTS_fld_FIRSTNAME) = sFirstName
        .Fields(SBSTUDENTS_fld_LASTNAME) = sLastName
        .Fields(SBSTUDENTS_fld_USERDEFINEDID) = sStudentID
        .Fields(SBSTUDENTS_fld_BILLINGSTATUS) = sStatus
        .Fields(SBSTUDENTS_fld_GRADELEVEL) = sGradeLevel
        .Fields(SBSTUDENTS_fld_GENDER) = sGender
        .Fields(SBSTUDENTS_fld_SSN) = sSSN
        .Fields(SBSTUDENTS_fld_BIRTHDATE) = dtBirthDate
        .Fields(SBSTUDENTS_fld_RELIGION) = sReligion
        .Fields(SBSTUDENTS_fld_ETHNICITY) = sEthnicity

        .StaticEntryField(SBSTUDENTS_fld_BILLINGOPTION) = _
        staticentry_SBR_SB_STU_UseThisBillingOption_AssignToStudent

        With .Statements.Add
            .Fields(STATEMENT_fld_PAYERID) = 2
            .StaticEntryField(STATEMENT_fld_STATEMENTTYPE) = _
            staticentry_StatementCharges_AllCharges
        End With

        'This needs to be done if you want to override warning rule given regarding the _
        'primary address if one does not exist
```

Adding a student record, continued (page 2 of 2)

```
Set oWarningRule = oStudent
        oWarningRule.OverrideWarning(SBStudent_Warning_PrimaryAddress) = True

        .Save
        .CloseDown
End With
    Set oStudent = Nothing
    Set oWarningRule = Nothing
End Sub
```

Student Billing Reports Samples

This section contains code samples for creating applications you can use with ***Student Billing*** reports. Samples include creating aged accounts receivable, open item, deposit list, and invoice reports.



Creating an Aged Accounts Receivable Report

With the following code sample, you can create an aged accounts receivable report.

```
Public Sub Create_SB_AgedReceivable_Report()
    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData
    Set oReport = goFE_Services.CreateReportInstance(bbrep_SB_AgedAccountsReceivable)
    Set oMetaData = oReport
    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Aged Receivable Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With
    With oMetaData

        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_BaseAgingOn_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_TODAY

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_TranDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = _
        bbDATE_LASTYEAR

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_DueDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_LASTYEAR

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_PostDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_LASTYEAR

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_IncludeBalance).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_BalanceAmount).Fields(REPORTPARAMETERVALUES_fld_CURRENCY) = 0
```

Creating an Aged Accounts Receivable Report, continued (page 2 of 2)

```
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_BalancePeriod).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = -1

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_SBAgedAR_ReduceBalance).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 1

'Set the various Filters
'Set the Charge Filter
With .PropertyDataObject(bbrep_Offset_Filters, bbrep_FilterParameter_FilterValues, _
            valuenumber:=1, ValueSet:=CStr(bbFilterType_SB_Charges))
            .Fields(FILTERS_fld_INCLUDEOPTION) = bbFilterIncludeOption_Range
            .Fields(FILTERS_fld_ACTION) = bbFilterAttributeActionType_Include
            .Fields(FILTERS_fld_FROMID) = 1
            .Fields(FILTERS_fld_TOID) = 10
        End With
    End With
    With oReport
        .Save
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating a Deposit Report

With the following code sample, you can create a deposit report.

```
Public Sub Create_SB_DepositReport_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Services.CreateReportInstance(bbrep_Common_DepositReport)
    Set oMetaData = oReport

    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Deposit List Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With
    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbrep_DepositReports_DepositDateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbrep_DepositReports_EnteredOnDateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbrep_DepositReports_PostDateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
       bbrep_DepositReports_DepositNumberFrom).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 2

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbrep_DepositReports_DepositNumberTo).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 5

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbrep_DepositReports_IncludeAmounts).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) _
        = True
```

Creating an Deposit Report, continued (page 2 of 2)

```
        .PropertyDataObject(bbrep_OffSet_ReportSpecific,_
        bbrep_DepositReports_IncludeAmountsValue).Fields _
        (REPORTPARAMETERVALUES_fld_NUMBER) = 100

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbrep_DepositReports_IncludeSB).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) = True
End With
    With oReport
        .Save
        .CloseDown
    End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Creating an Open Item Report

With the following code sample, you can create an open item report.

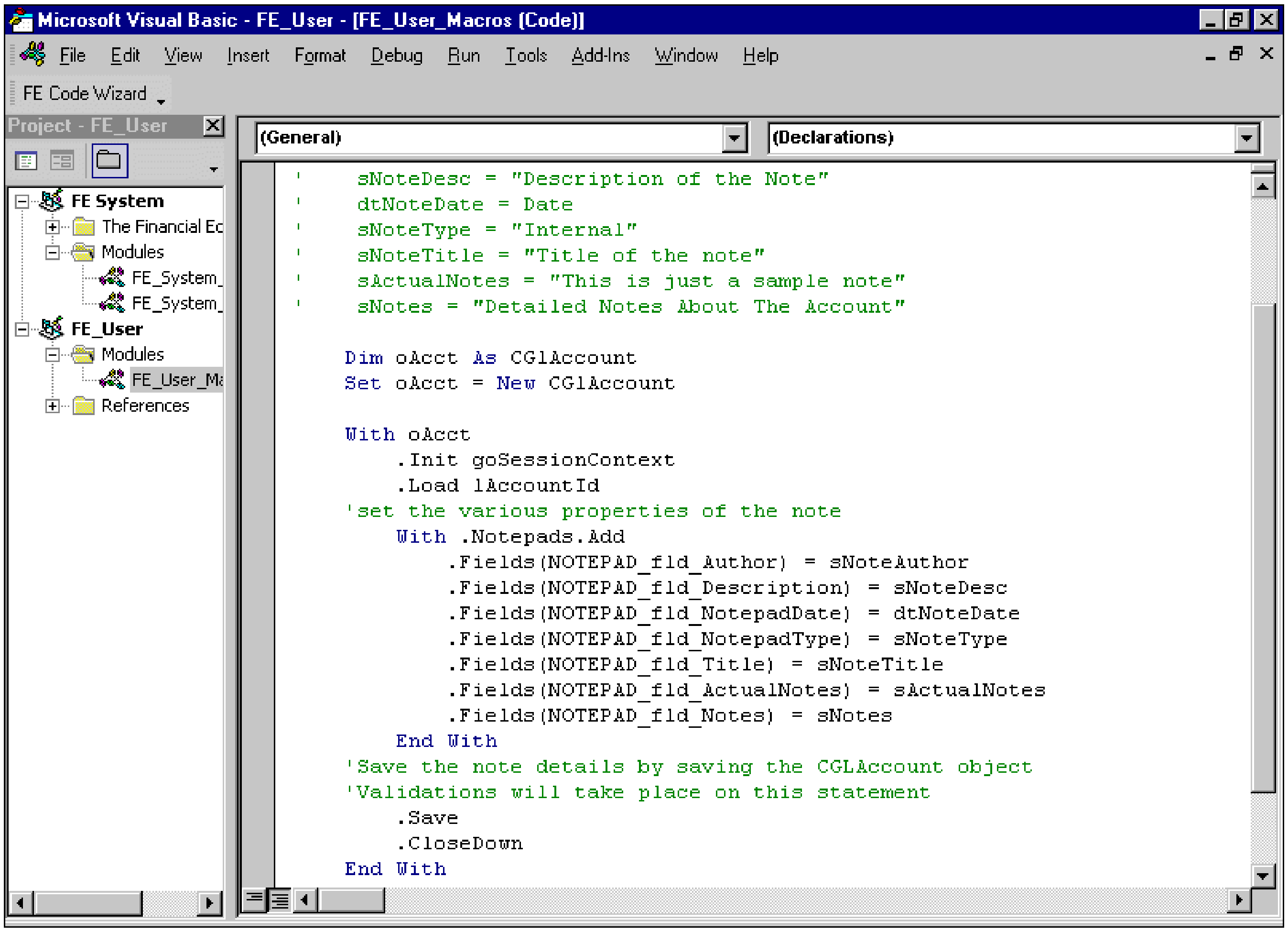
```
Public Sub Create_SB_OpenItem_Report()

    Dim oReport As IBBReportInstance
    Dim oMetaData As IBBReportMetaData

    Set oReport = goFE_Services.CreateReportInstance(bbrep_SB_OpenItemReport)
    Set oMetaData = oReport
    With oReport
        .Init goSessionContext

        'Set the name of the report
        .Property(REPORTPARAMETERNAMES_fld_NAME) = "Open Item Report Created From API"

        'Description of the report
        .Property(REPORTPARAMETERNAMES_fld_DESCRIPTION) = "Fields set from the API"

        'Can others execute this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSEXECUTE) = True

        'Can others modify this report ?
        .Property(REPORTPARAMETERNAMES_fld_OTHERSMODIFIY) = False
    End With
    With oMetaData
        'Set the report specific criteria
        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_OpenItem_ChargesOpenAsOf_DateType).Fields _
        (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_TODAY

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_OpenItem_BaseOpenDateOn).Fields(REPORTPARAMETERVALUES_fld_NUMBER) = 2

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_OpenItem_TransactionDate_DateType).Fields _
        (REPORTPARAMETERVALUES_fld_NUMBER) = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_OpenItem_PostDate_DateType).Fields(REPORTPARAMETERVALUES_fld_NUMBER) _
        = bbDATE_ALLDATES

        .PropertyDataObject(bbrep_OffSet_ReportSpecific, _
        bbRep_OpenItem_IncludeUnapplied).Fields(REPORTPARAMETERVALUES_fld_BOOLEAN) _
        = False
    End With
    With oReport
        .Save
        .CloseDown
```

Creating an Open Item Report, continued (page 2 of 2)

```
End With
    Set oReport = Nothing
    Set oMetaData = Nothing
End Sub
```

Common Samples

This section contains code samples for creating API applications you can use to control information in more than one ***Financial Edge*** program. Samples include code for creating notes, adding table entries, entering award amounts from grantors, creating invoices, defining multiple accounts, and managing and viewing reports.



```
'    sNoteDesc = "Description of the Note"
'    dtNoteDate = Date
'    sNoteType = "Internal"
'    sNoteTitle = "Title of the note"
'    sActualNotes = "This is just a sample note"
'    sNotes = "Detailed Notes About The Account"

    Dim oAcct As CGlAccount
    Set oAcct = New CGlAccount

    With oAcct
        .Init goSessionContext
        .Load lAccountId
    'set the various properties of the note
        With .Notepads.Add
            .Fields(NOTEPAD_fld_Author) = sNoteAuthor
            .Fields(NOTEPAD_fld_Description) = sNoteDesc
            .Fields(NOTEPAD_fld_NotepadDate) = dtNoteDate
            .Fields(NOTEPAD_fld_NotepadType) = sNoteType
            .Fields(NOTEPAD_fld_Title) = sNoteTitle
            .Fields(NOTEPAD_fld_ActualNotes) = sActualNotes
            .Fields(NOTEPAD_fld_Notes) = sNotes
        End With
    'Save the note details by saving the CGLAccount object
    'Validations will take place on this statement
        .Save
        .CloseDown
    End With
```

Using Global Variables

The following code sample illustrates the basics of initializing and closing global variables.

```
'The API and VBA samples use global variables like goFE_API, goFE_Service,
'goCodeTablesServer, goSessionContext
'They are defined and used as follows:
Public goFE_Service As IBBUtilityCode
Public goFE_API As FE_API
Public goSessionContext As IBBSessionContext
Public goCodeTablesServer As CCodeTablesServer

'Function to log into API and create Global variables
Public Function LoginToAPI(Optional ByVal sUsername As String, Optional ByVal spwd As
                           String) As Boolean
    Dim bInit As Boolean

    Set goFE_API = New FE_API
    goFE_API.SignOutOnTerminate = True

    If Len(sUsername) And Len(spwd) Then
        bInit = goFE_API.Init("", sUsername, spwd)
    Else
        bInit = goFE_API.Init("", "supervisor", "admin")
    End If

    If bInit Then
        Set goSessionContext = goFE_API.SessionContext
        Set goFE_Service = New FE_Services
        goFE_Service.Init goSessionContext
        Set goCodeTablesServer = New CCodeTablesServer
        goCodeTablesServer.Init goSessionContext
    End If
    LoginToAPI = bInit
End Function

'Function to release Global variables
Public Sub ReleaseGlobalObjects()
    If Not goFE_Service Is Nothing Then
        goFE_Service.CloseDown
        Set goFE_Service = Nothing
    End If

    If Not goCodeTablesServer Is Nothing Then
        goCodeTablesServer.CloseDown
        Set goCodeTablesServer = Nothing
    End If

    If Not goSessionContext Is Nothing Then
        Set goSessionContext = Nothing
    End If
    If Not goSessionContext Is Nothing Then
        Set goSessionContext = Nothing
```

(Continued- page 2 of 2)

```
    End If

    If Not goFE_API Is Nothing Then
        Set goFE_API = Nothing
    End If

End Sub
'NOTE: These variable have been defined with global scope for convenience. You can define
'    them with local or modular scope in your program/application.
```

Using Financial Edge as an Active Server Page

This code sample, frmAPSamples.frm, is located in the Help\Samples\Advanced_Samples\API\Samples\AP folder of the ***The Financial Edge*** installation directory. This sample demonstrates how you can access ***The Financial Edge*** as a server and use its functionality through your browser. When using ***The Financial Edge*** as a server, you should:

- Have *The API* installed on your server machine.
- Set the optional parameter ‘lAppMode’ to “amServer” or “1” when you init the API. The default is “amStandAlone”. This parameter tells ***The Financial Edge*** whether it is being run as a server or a standalone.
- Add the following line above the <head> tag to include the type library. This enables you to use the constant names and Intellisense.

  ```
  <!--METADATA TYPE="TypeLib" FILE="C:\program Files\The FinancialEdge\typelib\ _
  bbafnapi7.tlb" -->
  ```

  If you don’t know the path of the Typelib, you can specify the uuid parameter with the GUID inside curly brackets {}.
- Supply the username, password, and database number in the init. The API cannot initialize without these parameters, and it does not ask for them at a later stage (as in a standalone installation).
- Make sure no one logs into ***The Financial Edge*** on the server machine using the same username and password you use.
- Ensure SQL Server is kept running and connected to ***The Financial Edge*** database on the server machine. Failure to do so will result in performance lag.
- Use the FE_ComHelper.dll if you are using version IIS4. IIS versions 5 and higher permit casting objects to their supported interfaces, but IIS4 does not. To support this in IIS4, we have provided a provided a helper .dll called FE_ComHelper. The cQueryInteface class in FE_ComHelper has a method called getInterface which takes the GUID of an interface and object of a Class as its parameters. The GUID constants are available in the object browser (i.e. IBBTopObject_Guid). The getInterface function casts your object to the specified interface and returns it as the function value. (See the ASP Sample for details.)
- Use Server.CreateObject ("DllName.ClassName") to create an object of a class. All ***The Financial Edge*** objects should be available from the AFNAPI7.dll. You can also use ***The Financial Edge*** Services.GetProgID... functions to return the correct GUIDs.
- Access the SessionContext from the API object.
- Make sure the ASP account, usually "IUSR_machinename", has full rights to the <***The Financial Edge*** installation directory>\SysDB folder.

Adding a Note to a Record

This sample illustrates adding a notepad to an account, but you can alter this code to add notepads to any parent object that supports notes.

```
Friend Sub createNote(ByVal lAccountId As Long, ByVal sNoteAuthor As String, _
                            ByVal sNoteDesc As String, ByVal dtNoteDate As Date, _
                            ByVal sNoteType As String, ByVal sNoteTitle As String, _
                            ByVal sActualNotes As String, ByVal sNotes As String)

    'sNoteAuthor = "John Wright"
    'sNoteDesc = "Description of the Note"
    'dtNoteDate = Date
    'sNoteType = "Internal"
    'sNoteTitle = "Title of the note"
    'sActualNotes = "This is just a sample note"
    'sNotes = "Detailed Notes About The Account"

    Dim oAcct As CGlAccount
    Set oAcct = New CGlAccount

    With oAcct
        .Init goSessionContext
        .Load lAccountId
    'set the various properties of the note
        With .Notepads.Add
            .Fields(NOTEPAD_fld_Author) = sNoteAuthor
            .Fields(NOTEPAD_fld_Description) = sNoteDesc
            .Fields(NOTEPAD_fld_NotepadDate) = dtNoteDate
            .Fields(NOTEPAD_fld_NotepadType) = sNoteType
            .Fields(NOTEPAD_fld_Title) = sNoteTitle
            .Fields(NOTEPAD_fld_ActualNotes) = sActualNotes
            .Fields(NOTEPAD_fld_Notes) = sNotes
        End With
    'Save the note details by saving the CGLAccount object
    'Validations will take place on this statement
        .Save
        .CloseDown
    End With
    Set oAcct = Nothing
End Sub
```

Finding a Code Table Entry ID

This code sample returns a code table entry ID for a specified code table.

```
'This function returns the ID for the CodeTableEntry in the CodeTable specified by its ID
'(lCodeTableType)
'e.g. ctnumGLClass or ctnumFADisposalMethod

Friend Function getIdFromCodeTable( _
                              ByVal lCodeTableType As Long, ByVal sEntry As String) As Long
    Dim oTableEntries As CTableEntries
    Set oTableEntries = New CTableEntries

    Dim oTableEntry As CTableEntry

    With oTableEntries
        .Init goSessionContext, lCodeTableType, True

        For Each oTableEntry In oTableEntries
            If oTableEntry.Fields(tableentry_fld_DESCRIPTION) = sEntry Then
                getIdFromCodeTable = oTableEntry.Fields(tableentry_fld_TABLEENTRIESID)
                Exit For
            End If
        Next
    End With
    oTableEntry.CloseDown
    oTableEntries.CloseDown
    Set oTableEntry = Nothing
    Set oTableEntries = Nothing
End Function
```

Finding an Attribute Value ID

This code sample returns an attribute value ID for a specified attribute type.

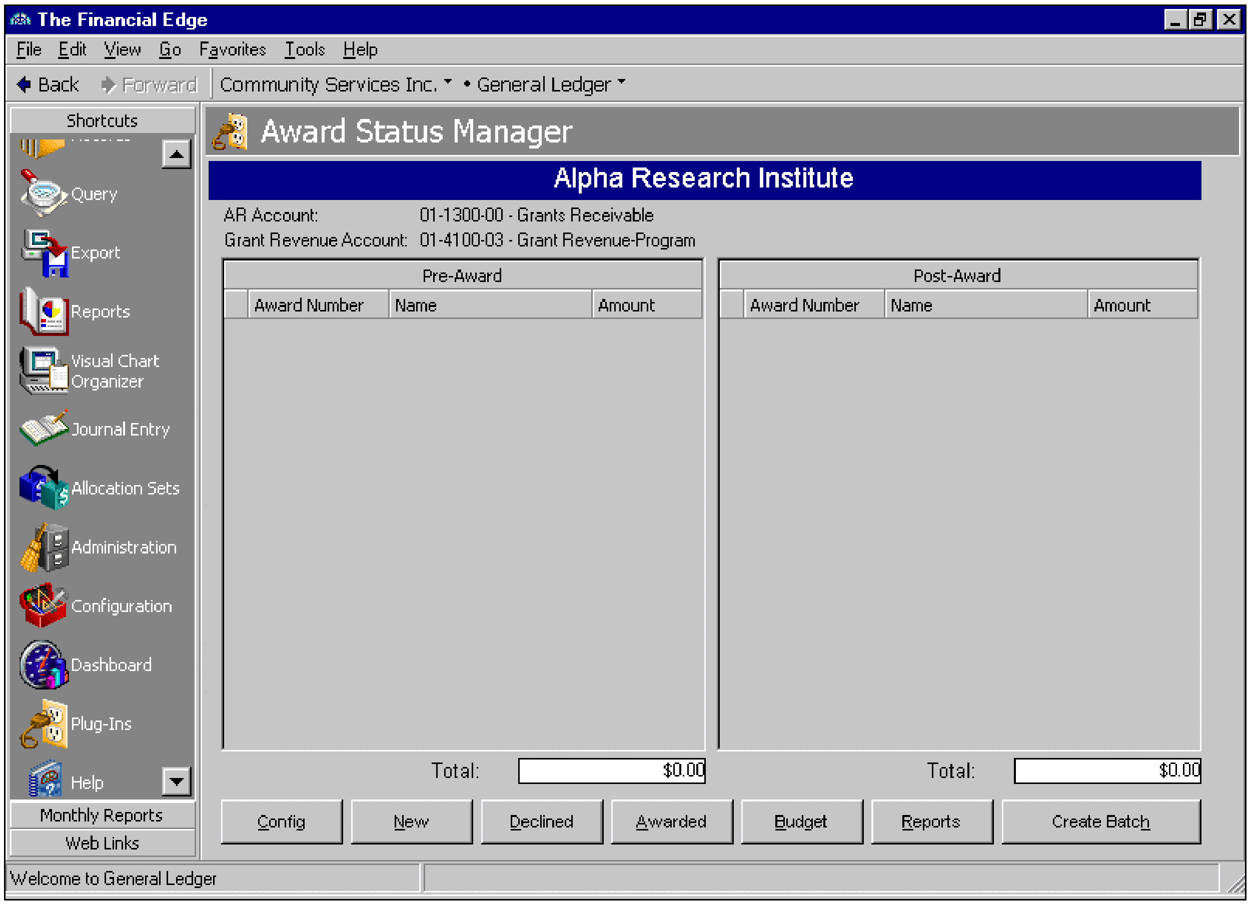
```
'This function returns the ID of the Attribute Value of the Attribute Type whose ID is
'specified by lAttributeType

 Friend Function getAttributeTypeId(ByVal sAttribDesc As String, _
                                    ByVal lAttributeType As Long) As Long
    Dim lAttribTypeId As Long
    Dim oAttribServer As CAttributeTypeServer
    Set oAttribServer = New CAttributeTypeServer

    With oAttribServer
        .Init goSessionContext, lAttributeType
        lAttribTypeId = .getAttributeTypeId(sAttribDesc)
        .CloseDown
    End With
    Set oAttribServer = Nothing
    getAttributeTypeId = lAttribTypeId
End Function
```

Using the Award Status Manager

The Award Status Manager is a sample plug-in that illustrates how you can create custom forms to enter information into ***The Financial Edge***. With this plug-in, you can add, view, and modify the status of grant awards from one place. You can add grants for which you have applied, log the expected award amount, and enter the actual award amount as a project record when you receive the grant. Using buttons at the bottom of the Award Status Manager page, you can also easily add budget information and create reports and batches. To install the Award Plug-in, browse to the Financial Edge Help/Samples/Advanced_Samples/API/Plug-Ins/AwardPlugin folder. Copy the AwardPlugin.dll, AwardPlugin.mdb, and docAwardPlugin.vbd files, then paste them into The Financial Edge/Plugins folder.

➢ Using the Award plug-in

1. To use the Award plug-in, on the navigation bar of ***The Financial Edge***, click Plug-Ins. The Plug-Ins page appears.
2. Click **Award Status Manager**. The Award Status Manager screen appears.



Alpha Research Institute

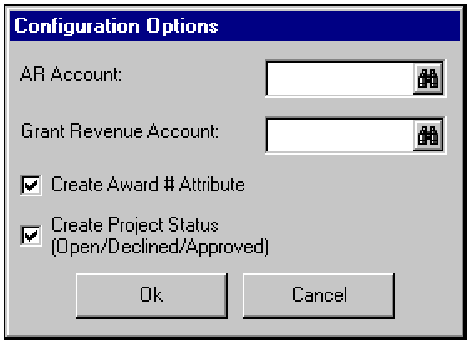
AR Account: 01-1300-00 - Grants Receivable
Grant Revenue Account: 01-4100-03 - Grant Revenue-Program

Pre-Award			Post-Award		
Award Number	Name	Amount	Award Number	Name	Amount

Total: $0.00 Total: $0.00

Welcome to General Ledger

3. Click **Config**. The Configuration Options screen appears.



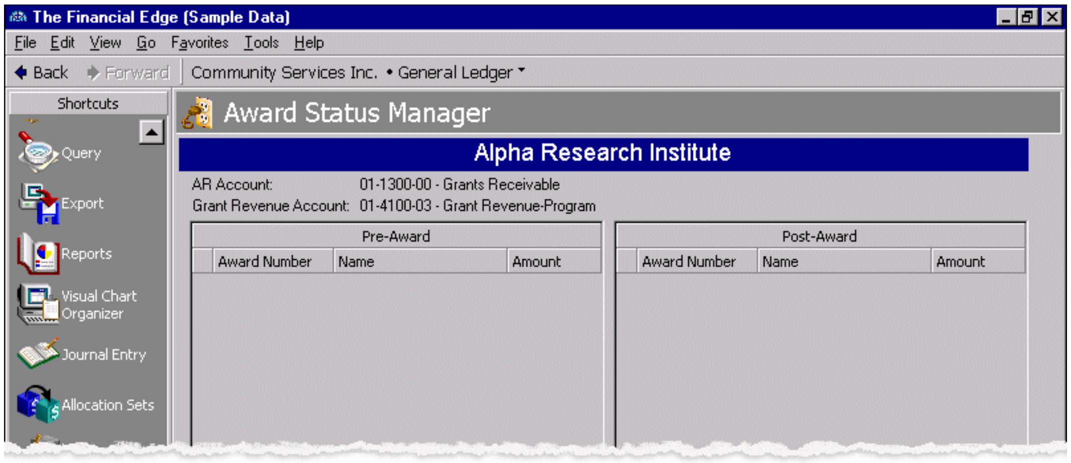
Configuration Options

AR Account:	
Grant Revenue Account:	

[✓] Create Award # Attribute

[✓] Create Project Status (Open/Declined/Approved)

4. In the **AR Account** field, enter an accounts receivable account number for the award.
5. In the **Grant Revenue Account** field, enter a revenue account number.
6. Mark **Create Award # Attribute** and **Create Project Status (Open/Declined/Approved)**. These checkboxes automatically create a project attribute with three possible field entries describing the status of the award.
7. Click **OK**. The Configuration Options screen closes and the Award Status Manager page appears with the new award information in the heading.



The Financial Edge (Sample Data)
File Edit View Go Favorites Tools Help
Back Forward Community Services Inc. • General Ledger
Shortcuts
Query
Export
Reports
Visual Chart Organizer
Journal Entry
Allocation Sets

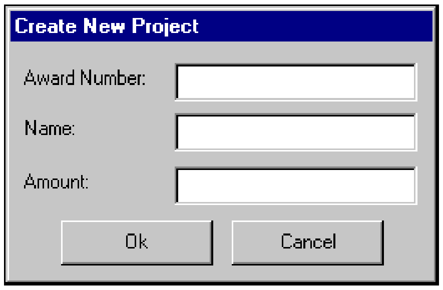
Award Status Manager

Alpha Research Institute

AR Account:	01-1300-00 - Grants Receivable
Grant Revenue Account:	01-4100-03 - Grant Revenue-Program

Pre-Award			Post-Award		
Award Number	Name	Amount	Award Number	Name	Amount

8. To change the configuration for this award, click **Config**.
9. After you have configured the Award plug-in, to create a new award, click **New**. The Create New Project screen appears.



Create New Project

Award Number:	
Name:	
Amount:	

10. In the **Award Number** field, enter a number for the award.
11. In the **Name** field, enter a project name.

12. In the **Amount** field, enter the project amount.
13. To create a new ***Financial Edge*** project, click **OK**. A new ***General Ledger*** project record opens with the information for the new project. For information about adding project records, see ***The Financial Edge*** help file.
14. After you add information to the new project, click **Save and Close** to save the project and return to the Award Status Manager page.
15. All projects with an open status appear in the **Pre-Award** grid. If a grant is not awarded, you can select the project and click **Declined** to remove the project from the list. If the grant is awarded, select the project and click **Awarded** to move the project to the **Post-Award** grid.
16. To add budget information, select a project and click **Budget**.
17. To view an income statement report for a project, select a project and click **Reports**.
18. To create batch entries for projects, click **Create Batch**. ***The Financial Edge*** creates journal entries for all the projects, unless they have already been entered. Pre-awards are entered as encumbrances, and declined awards are entered as reversed encumbrances.

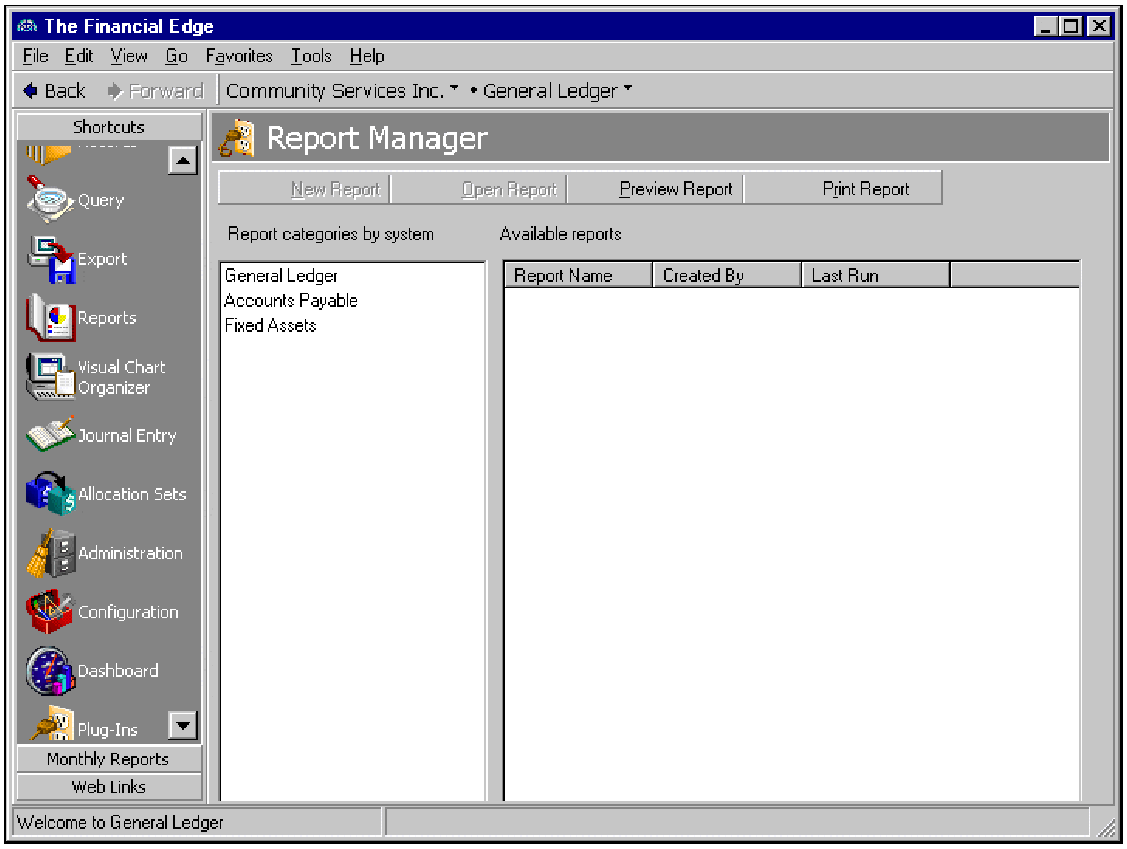
Using the Report Manager

The Report Manager is a plug-in you can use to manipulate all ***Financial Edge*** reports from a single location without having to toggle between programs. With the Report Manager you can create a new report, open an existing report, preview, or print any report in ***The Financial Edge***. To install the Report Manager, browse to the Financial Edge Help/Samples/Advanced_Samples/API/Plug-Ins/ReportManager folder. Using the FEReportManager.vdb file, in *Visual Basic*, select **File**, **Make FEReportManager.dll**. After *Visual Basic* creates the .dll file, copy both the FEReportManager.dll and docReportManager.vbd files, then paste them into The Financial Edge/Plugins folder.

➢ **Using the Report Manager plug-in**

1. To use the Report Manager plug-in, on the navigation bar of ***The Financial Edge***, click **Plug-Ins**. The Plug-Ins page appears.

2. Click **Award Status Manager**. The AwardPlugin screen appears.



General Ledger
Accounts Payable
Fixed Assets

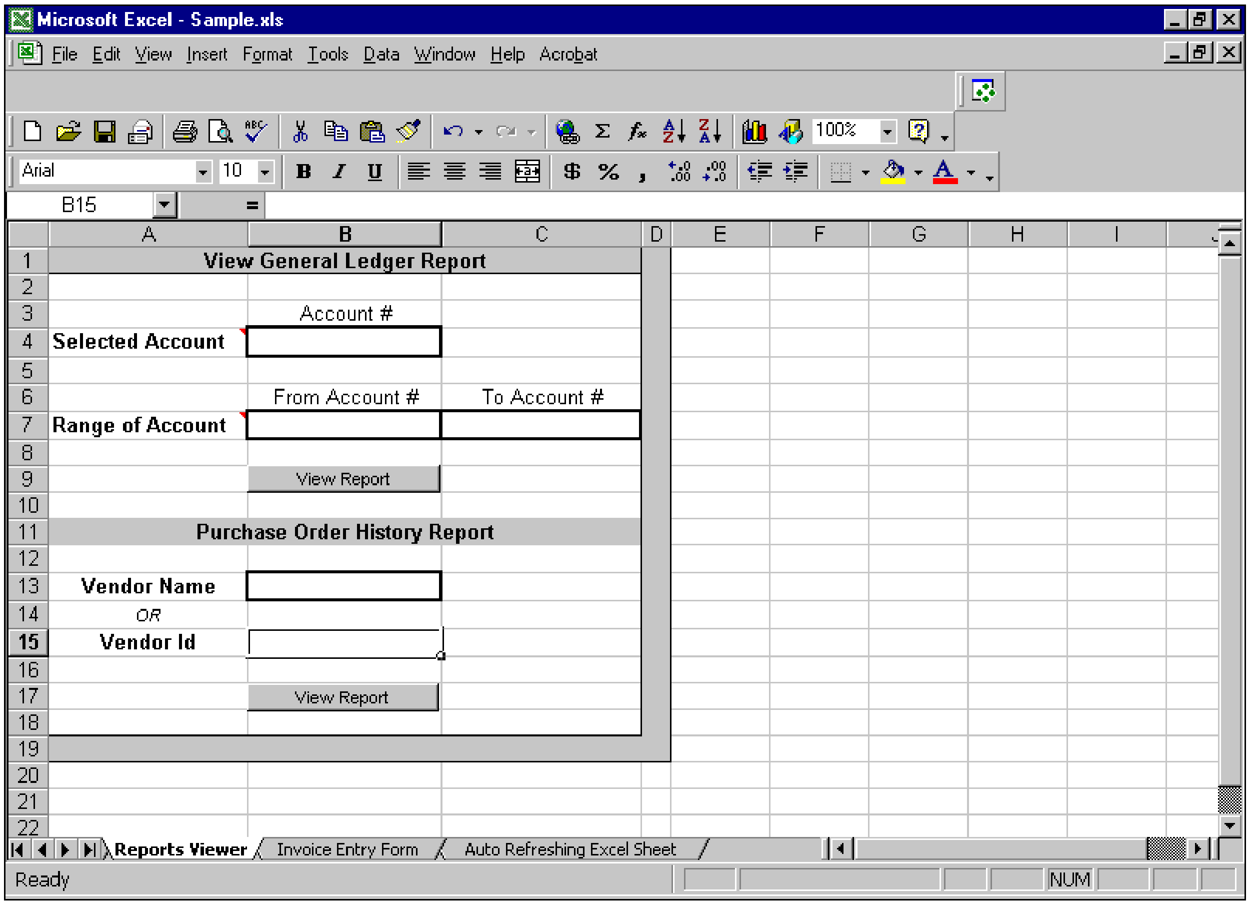
Welcome to General Ledger

3. In the grid, in the **Report categories by system** column, double-click a program name to expand the list of reports.
4. From the expanded list, select a report type. Existing reports of that type appear in the **Available Reports** column.
5. To create a new report, click **New**. The New <Type> Report screen appears. For information about creating specific reports, see ***The Financial Edge*** help file.
6. To open an existing report, in the **Available Reports** column, select a report and click **Open**.
7. To preview a report, select a report and click **Preview Report**.
8. To print a report, select a report and click **Print Report**.

Using the Report Viewer

The Report Viewer is a plug-in that demonstrates how you can integrate ***The Financial Edge*** and Microsoft *Office* to open a ***General Ledger*** report or a purchase order history report from an *Excel* spreadsheet. To use the Report Viewer, browse to the Help/Samples/Advanced_Samples/API/Samples/SamplesinXLS folder and copy the Samples.xls file. Then, paste Samples.xls in the Financial Edge/Plug-ins folder. Open the Samples.xls and select the Reports Viewer tab. Cells with red flags contain helpful instructions for entering information in the spreadsheet.

To view a ***General Ledger*** report, enter an account number or a range of accounts, then click **View Report**. To view a purchase order report, enter a vendor name or vendor ID, then click **View Report**.


Microsoft Excel - Sample.xls

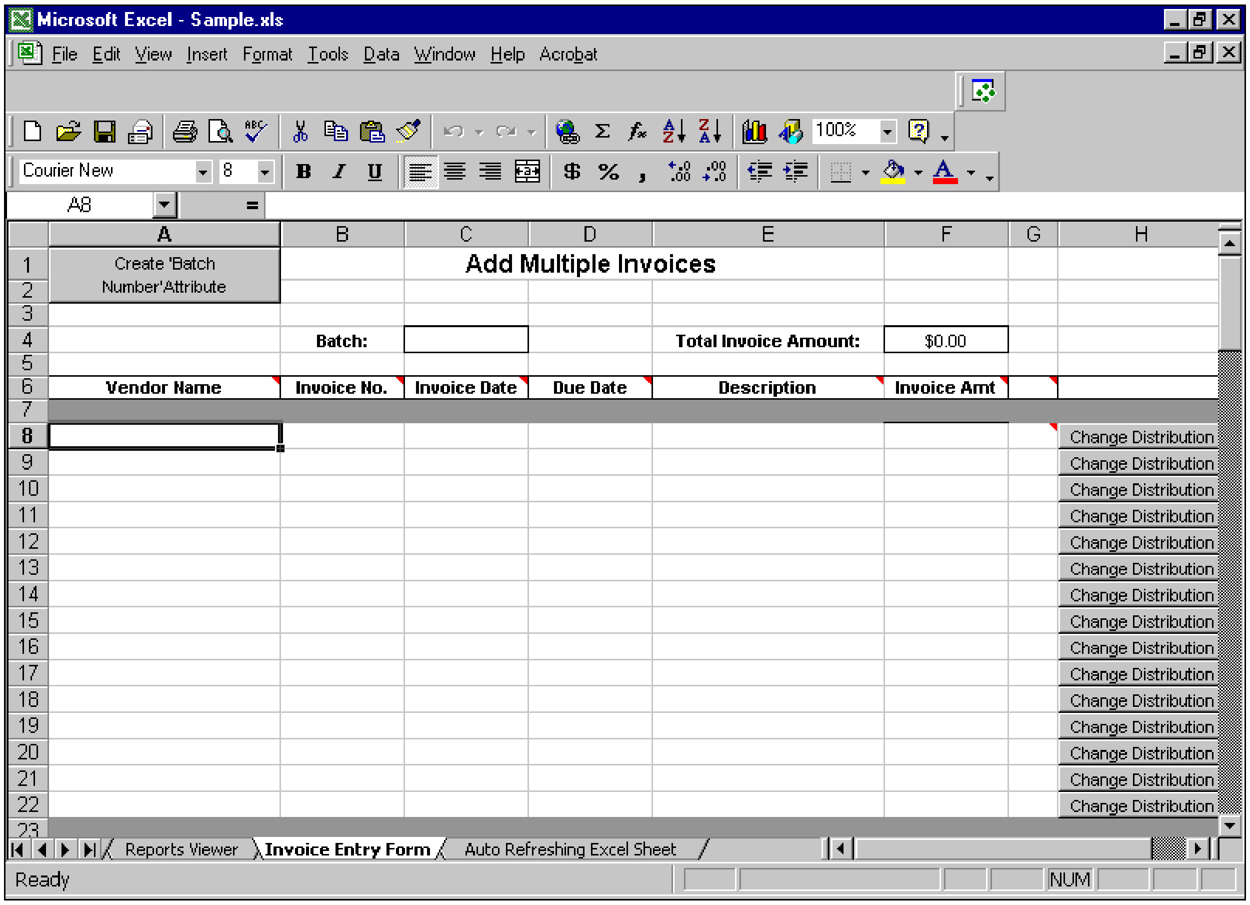
	A	B	C
1	View General Ledger Report		
2			
3		Account #	
4	Selected Account		
5			
6		From Account #	To Account #
7	Range of Account		
8			
9		View Report	
10			
11	Purchase Order History Report		
12			
13	Vendor Name		
14	OR		
15	Vendor Id		
16			
17		View Report	

File Edit View Insert Format Tools Data Window Help Acrobat
Arial 10 100%
B15
Reports Viewer
Invoice Entry Form
Auto Refreshing Excel Sheet
Ready
NUM

Using the Invoice Entry Form

The Invoice Entry form, another example of the integration possible between *Excel* and ***The Financial Edge***, is located in the Samples.xls file in the Advanced_Samples/API/Samples/SamplesinXLS folder. With this form, you can speed the process of entering multiple invoices by entering up to 15 invoices on a single *Excel* spreadsheet and then saving them to ***The Financial Edge*** database with just one click. To use the Invoice Entry Form, copy and paste the Samples.xls file into the Financial Edge/Plug-ins folder. Then, open the file with *Excel* and select the Invoice Entry Form tab.

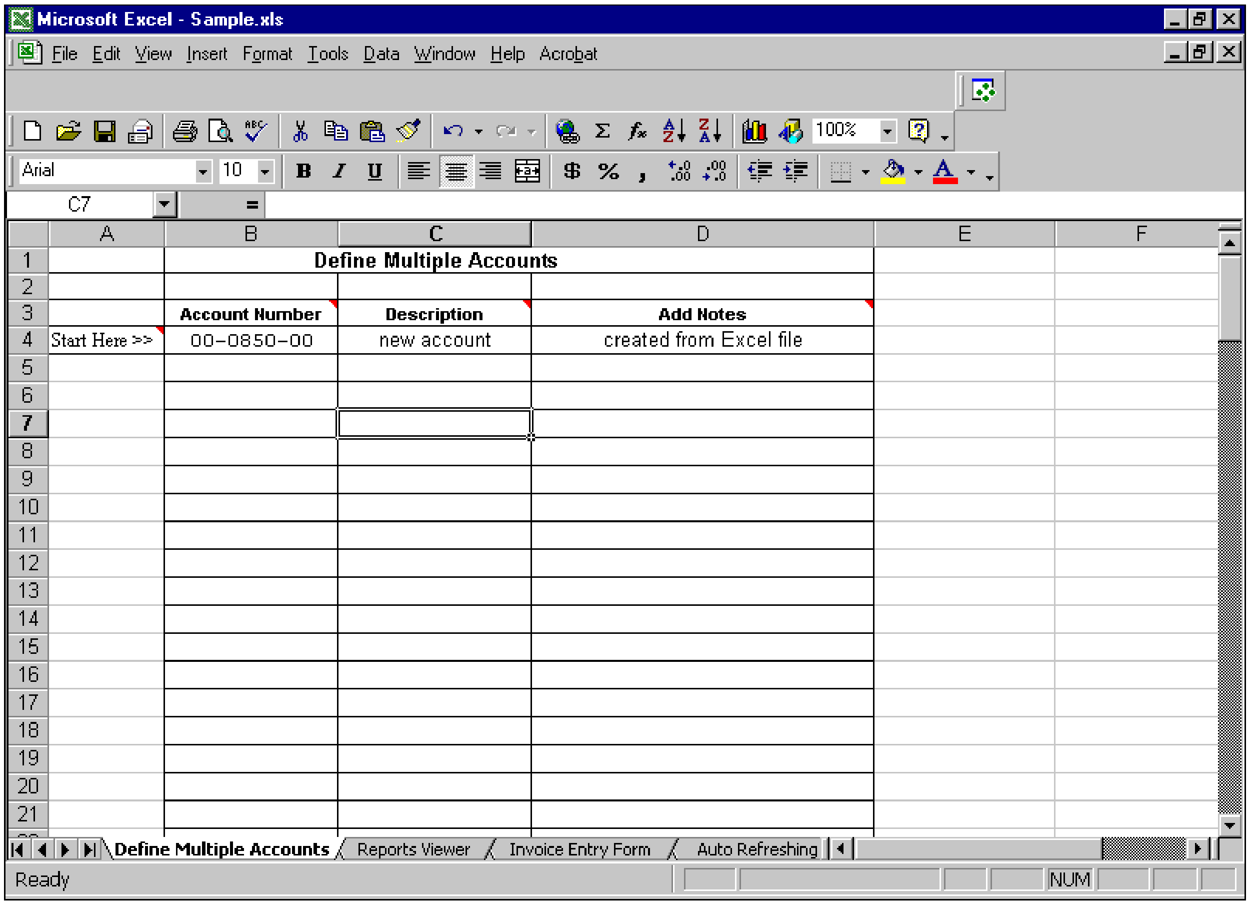
To distribute the invoices, you can either use the vendor's default distribution, or to create a new distribution, select a row and click **Change Distribution**. To save the invoices to ***The Financial Edge*** database and clear the spreadsheet, click **Save Invoices**. To delete all invoices without saving, click **Clear Cells**.


Microsoft Excel - Sample.xls
File Edit View Insert Format Tools Data Window Help Acrobat
Courier New
8
100%
A8
=

	A	B	C	D	E	F	G	H
1	Create 'Batch Number'Attribute		Add Multiple Invoices					
2								
3								
4		Batch:			Total Invoice Amount:	$0.00		
5								
6	Vendor Name	Invoice No.	Invoice Date	Due Date	Description	Invoice Amt		
7								
8								Change Distribution
9								Change Distribution
10								Change Distribution
11								Change Distribution
12								Change Distribution
13								Change Distribution
14								Change Distribution
15								Change Distribution
16								Change Distribution
17								Change Distribution
18								Change Distribution
19								Change Distribution
20								Change Distribution
21								Change Distribution
22								Change Distribution
23								

Defining Multiple Accounts

With the Define Multiple Accounts form, you can increase efficiency by creating multiple accounts for ***The Financial Edge*** from an *Excel* worksheet. To use the Define Multiple Accounts form, browse to the Financial Edge Help/Samples/Advanced_Samples/API/Samples/SamplesinXLS folder and copy the Samples.xls file. Then, paste Samples.xls in the Financial Edge/Plug-ins folder. Open the Samples.xls file with *Excel* and select the Define Multiple Accounts tab. Cells with red flags contain helpful instructions for entering information in the spreadsheet.



	A	B	C	D
1		Define Multiple Accounts		
2				
3		Account Number	Description	Add Notes
4	Start Here >>	00-0850-00	new account	created from Excel file

After you enter an account number and description (notes are optional), click a cell in the **Add Notes** column and press **TAB**. A New Account screen appears with the information you entered. You can enter additional information about the account, or, to save the account and close the New Account screen, click **Save and Close**.

VBA Samples

The Financial Edge includes several VBA sample programs installed in The_Financial_Edge\Help\Samples\VBA folder. These samples basically consist of two different samples — a notepad sample and a spelling checker application you can use on notepads.

The Financial Edge includes the following VBA code samples:

Sample	Format	Description
Notepad	*Visual Basic 6.0*	This macro creates notepads without having to open a parent object record.
SpellCheck	*Visual Basic 6.0*	This macro checks for spelling errors on notepads.

Additionally, this guide contains VBA code samples for creating new business rules and macros.

Validating Dates

This code sample verifies that all dates entered on an object fall within two years of today's date.

Instructions:

1. In a VBA module, add the ValidDates function.
2. Add the following code to the BeforeSave event of the object you want to validate:

```
bCancel = Not ValidDates(oFAAsset)
'Change to pass in the appropriate object
```

The following code sample illustrates using the ValidDates function to require a date within the next two years:

```
Public Function ValidDates(oDataobject As IBBDataObject) As Boolean
'Can add to the BeforeSave event of any object to validate that all object dates are within
'2 years of today.

    Dim oBBMetaField As IBBMetaField
    Set oBBMetaField = oDataobject

    Dim l As Long
    Dim sTemp As String
    Dim bValid As Boolean

    bValid = True

    For l = 1 To oBBMetaField.Count 'Really Field Count
        With oBBMetaField
        If .FormatDescriptor(l) = fmtDATEMDY Then
                If oDataobject.FieldIsDirty(l)Then
                    sTemp = RTrim$(oDataobject.Fields(l))
                    If Len(sTemp) > 0 Then
                        If Abs(DateDiff("yyyy", sTemp, Date$)) > 2 Then
                        'If the date is not within 2 years raise an error
                            If MsgBox("The field '" & .DisplayText(l) & ' is not within 2 _
                                years of today. Save the record anyway?", _
                                    vbYesNo) = vbNo Then
                                bValid = False
                                Exit For
                            End If
                        End If
                    End If
                End If
            End If
        End With
    Next l
    Set oBBMetaField = Nothing

    ValidDates = bValid
End Function
```

Viewing Query Results in HTML

The following code sample uses query results to create an HTML page you can access from the **Help** menu:

```
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
    (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, _
     ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) _
       As Long

Public Sub PublishToWeb(o As IBBQueryRow)
    On Error GoTo eh
        If o.BOF Then
            Static lFil As Long
            If lFil <> 0 Then
                Close lFil
            End If

            lFil = FreeFile
            Open "c:\WebQuery.htm" For Output As #lFil

            Print #lFil, "<HTML>"
            Print #lFil, "<HEAD>"
            Print #lFil, "</HEAD>"
            Print #lFil, "<BODY>"
            Print #lFil, "<XML ID=QueryData>"
            Print #lFil, "<QUERYROWS>"

        ElseIf o.EOF Then

        Print #lFil, "</QUERYROWS>"
        Print #lFil, "</XML>"
        Print #lFil, "<SPAN style='font-family:tahoma;font-size:20px'>Query Results Page_
          </SPAN><HR>"

        Print #lFil, "<A ID=PrevPage style='color:blue' href=""#"">&lt;&lt; Prev Page</A>"
        Print #lFil, "&nbsp;&nbsp;&nbsp;&nbsp;"
        Print #lFil, "<A ID=NextPage style='color:blue' href=""#"">Next Page >></A>"
        Print #lFil, "<SCRIPT Language=VBSCRIPT>"
        Print #lFil, "Sub NextPage_OnClick()"
        Print #lFil, "tblData.nextPage"
        Print #lFil, "End Sub"
        Print #lFil, "Sub PrevPage_OnClick()"
        Print #lFil, "tblData.previousPage"
        Print #lFil, "End Sub"
        Print #lFil, "</SCRIPT>"
        Print #lFil, "<TABLE Border=1 ID=tblData DATAPAGESIZE=15 border=1 _
          style='Font - family: tahoma' DATASRC=#QueryData>"
        Print #lFil, "<THEAD style='background-color:maroon;color:white'>"

        Dim l As Long

        For l = 1 To o.FieldCount
            Print #lFil, "<TH>" & o.FieldName(l) & "</TH>"
```

(Continued- page 2 of 2)

```
        Next l

        Print #lFil, "</THEAD>"
        Print #lFil, "<TR>"
        Dim sFld As String
        For l = 1 To o.FieldCount
            sFld = Trim$(UCase$(o.FieldName(l)))
            Print #lFil, "<TD valign=Top><SPAN DATAFLD=""" & fixField(sFld) & """>_
              </SPAN></TD>"
        Next l

        Print #lFil, "</TR>"
        Print #lFil, "</TABLE>"
        Print #lFil, "</BODY>"
        Print #lFil, "</HTML>"

        Close lFil

        On Error Resume Next
        ShellExecute FE_Application.SessionContext.MainForm.hwnd, "Open", _
          "c:\WebQuery.htm", "", "c:\", 1
        On Error GoTo 0

    Else
        ' Write each row out
        Print #lFil, "<QUERYROW>"
        For l = 1 To o.FieldCount
            Print #lFil, "<" & fixField(o.FieldName(l)) & ">" & FixDataField(o.Field(l)) _
              & "</" & fixField(o.FieldName(l)) & ">"
        Next l
        Print #lFil, "</QUERYROW>"
    End If
Exit Sub

eh:
    MsgBox "Error: " & Err.Description
    Exit Sub
End Sub
Private Function fixField(ByVal s As String) As String
    Dim stemp As String
    stemp = UCase$(Trim$(Replace(s, " ", "_")))
    fixField = UCase$(Trim$(Replace(stemp, "/", "_")))
End Function

Private Function FixDataField(ByVal s As String) As String
    FixDataField = UCase$(Trim$(Replace(s, "&", "_")))
End Function
```

Calculating GST and PST Amounts

➢ With this sample, located in the Advanced Samples folder, you can calculate GST and PST tax rates for Australia and Canada. **Setting Up the Sample**

1. Before you can run this code, you must add two forms, Frm_Tax and Frm_InvoiceAmount, to your VBA code. You can locate these forms in the Financial Edge\Help\Advanced Samples\VBA\GST folder. To add the forms, in the Project Explorer, expand the FE System folder and select the Forms folder. Right-click and select **Import File**, then browse to the forms files.
2. Create the following macro (in FE_System_Macros) to setup your GST/PST values:

```
Public Sub DefineTax()
    frm_Tax.Show vbModal
End Sub
```

3. Add the code in FE_System_Object_Code to FE_System_Object_Code.
4. Add the following to the APInvoice_BeforeOpen Event:

```
Dim oAPRecord As CAPInvoice
Set oAPInvoice = oRecord
HandleGST oAPInvoice
Set oAPInvoice = Nothing
```

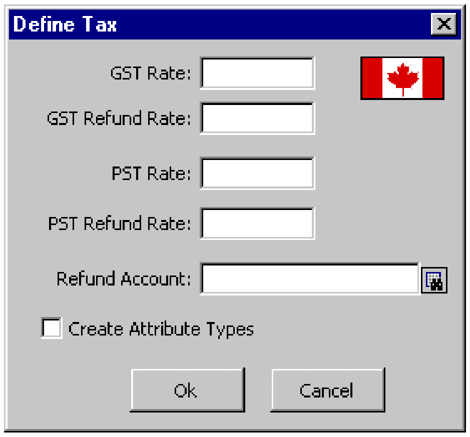
5. Copy the GST.MDB to the Plugins folder.
6. Add a standard code UIOpening and UIClosing Event:

```
Private Sub FE_Application_UIClosing(bCancel As Boolean)
    ReleaseFEGlobalObjects True
End Sub

Private Sub FE_Application_UIOpening()
    CreateFEGlobals FE_Application.SessionContext
End Sub
```

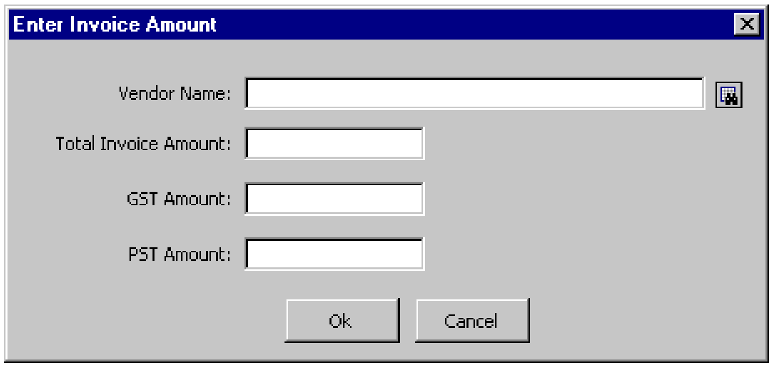
7. Add a reference to the Microsoft DAO.
8. To use the sample, see “Using the Sample” on page 274.

➢ Using the Sample

1. To define GST/PST amounts and a refund account, run the Define Tax macro.



2. To automatically calculate GST and PST amounts when adding new invoices, run the Enter Invoice Amount macro. When you enter a vendor name and invoice amount, the macro calculates the GST and PST amounts based on defined rates.



Sending a Warning Message for Large Invoices

With this macro, if a user creates an invoice exceeding $1000, the program automatically generates an email to your organization’s CFO.

```
'Add the following line to the APInvoice_AfterSave Event:
'DoBigInvoiceEmail oRecord

Public Sub DoBigInvoiceEmail(oRecord As Object)

    Dim oAPInvoice As CAPInvoice
    On Error GoTo ErrHandler
    Set oAPInvoice = oRecord

    If oAPInvoice.Fields(APINVOICES_fld_INVOICEAMOUNT) > 1000 Then
        Dim oOutlook As Outlook.Application
        Set oOutlook = CreateObject("Outlook.Application")

        Dim oMailItem As MailItem
        Set oMailItem = oOutlook.CreateItem(olMailItem)

        oMailItem.To = "CFO@YourOrganization.com"
        oMailItem.Subject = "Large Invoice Alert"

        Dim oApVendor As cAPVendor
        Set oApVendor = New cAPVendor
        oApVendor.Init goFE_Sessioncontext
        oApVendor.Load oAPInvoice.Fields(APINVOICES_fld_AP7VENDORSID), True
        Dim sTemp As String

        sTemp = "An invoice for " & oApVendor.Fields( _
                 APVENDORS_fld_VENDORNAME_FORDISPLAY) & " has just been added _
                 in the amount of " & Format$(oAPInvoice.Fields( _
                 APINVOICES_fld_INVOICEAMOUNT), "currency") & "."
        sTemp = sTemp & "    The invoice description is _
        '" & oAPInvoice.Fields(APINVOICES_fld_DESCRIPTION) & "'"
        oMailItem.Body = sTemp

        oMailItem.Display
        'Shows the email, so user can see it
        'oMailItem.Send
        'Sends the email without user intervention

        Set oMailItem = Nothing
        Set oOutlook = Nothing

        oApVendor.CloseDown
        Set oApVendor = Nothing
    End If
```

(Continued- page 2 of 2)

```
    Set oAPInvoice = Nothing

    On Error GoTo 0

    Exit Sub

ErrHandler:
    Dim sErr As String
    sErr = Err.Description

    On Error GoTo 0
    '< place your custom error handling code here >
    MsgBox "Error processing DoBigInvoiceEmail : " & sErr

    If Not oApVendor Is Nothing Then
        oApVendor.CloseDown
        Set oApVendor = Nothing
    End If

    Set oAPInvoice = Nothing

End Sub
```

Requiring Approval for Large Purchase Orders

This sample adds an action to a specified user if another users adds a purchase order exceeding a specific amount. You can use this security feature to require your CFO’s approval for large purchase orders.

Instructions:

1. Add the AddActionforPOApproval function to a VBA module
2. Add the following code to the APPurchaseOrder_BeforeSave event:

```
AddActionforPOApproval oAPPurchaseOrder
```

The following code sample illustrates requiring approval for purchase orders exceeding $500:

```
Public Sub AddActionforPOApproval(oAPPurchaseOrder As CAPPurchaseOrder)

    With oAPPurchaseOrder
        If .Fields(APPURCHASEORDERS_fld_PURCHASEORDERTOTAL) > 500 Then
            If goFE_Service.UserGetName(goSessionContext.CurrentUserID) = "Joe" Then
                Dim oVendor As cAPVendor
                Set oVendor = New cAPVendor
                oVendor.Init goFE_Sessioncontext
                oVendor.Load .Fields(APPURCHASEORDERS_fld_AP7VENDORSID)
                With oVendor
                    With .Actions.Add
                        .Fields(ACTIONS_fld_ACTIONDATE) = Date$
                        .Fields(ACTIONS_fld_ACTIONTYPE) = "Follow Up"
                        .Fields(ACTIONS_fld_ASSIGNEDTOID) = "supervisor" _
                                                   'goFE_UtilityCode.UserGetID("Supervisor")
                        .Fields(ACTIONS_fld_AUTOREMIND) = True
                        .Fields(ACTIONS_fld_REMINDUSERID) = _
                           .Fields(ACTIONS_fld_ASSIGNEDTOID)
                        .Fields(ACTIONS_fld_DESCRIPTION) = "PO Approval"
                        .Save
                    End With
                    .Save
                    .CloseDown
                End With
                Set ApVendor = Nothing
            End If
        End If
    End With

End Sub
```

Creating an Excel Chart from Query Results

This code sample creates an *Excel* chart from the results of a query.

```
' This macro can be assigned to a query or executed via the
' "Process VBA macro" option
' This code requires a reference to the Microsoft Office type libraries
' Note: for the Total Journal Fields, miscellaneous fields, and chart to work correctly,
' run this on a query with (Real Amount (credits multipled by -1)) and Journal in the
' first 2 fields of the Output

Public Sub DumpQueryToExcel(o As IBBQueryRow)

    If o.BOF Then
        Set moExcel = CreateObject("Excel.Application") 'Opens Excel
        moExcel.Visible = True

        moExcel.Workbooks.Add 'Add a new worksheet
        Set moWorksheet = moExcel.Worksheets(1)

        Dim lHeads As Long 'Fills the first row with
        'For lHeads = 1 To o.FieldCount - 1 'the field names from the query
        'moWorksheet.Cells(1, lHeads) = o.FieldName(lHeads)
        'Next lHeads
        moWorksheet.Cells(1, 1) = "Amount"
        moWorksheet.Cells(1, 2) = "Journal"

    ElseIf o.EOF Then

        ' Post Process Some Results In Excel
        With moWorksheet
            .Columns("A:E").EntireColumn.AutoFit
            .Columns("A:C").Select
        End With

        moExcel.Application.Selection.Sort Key1:=moWorksheet.Range("A2"), _
             Order1:=xlAscending, Header:=xlGuess, OrderCustom:=1, MatchCase:=False, _
             Orientation:=xlTopToBottom
        moWorksheet.Columns(1).NumberFormat = "$#,##0.00_)"

        moWorksheet.Cells(1, 4) = "Total 'Journal Entry'"
        moWorksheet.Cells(2, 4) = "Total 'Accounts Payable'"
        moWorksheet.Cells(3, 4) = "Total 'Allocation Management'"
        moWorksheet.Cells(4, 4) = "Total 'Cash Management'"
        moWorksheet.Cells(5, 4) = "Total 'Purchase Orders'"
        moWorksheet.Cells(7, 4) = "Total 'Student Billing'"
        moWorksheet.Cells(8, 4) = "Total 'Accounts Receivable'"
        moWorksheet.Cells(9, 4) = "Total 'Fixed Assets'"
        moWorksheet.Cells(10, 4) = "Total 'Cash Receipts'"
        moWorksheet.Cells(1, 5).Formula = "=SUMIF( _

          B2:B" & o.RowNum + 1 & ","""=Journal Entry""",A2:A" & o.RowNum + 1 & ")"
         moWorksheet.Cells(2, 5).Formula = "=SUMIF( _
```

(Continued- page 2 of 3)

```
             B2:B" & o.RowNum + 1 & ",""=Accounts Payable"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(3, 5).Formula = "=SUMIF( _
             B2:B" & o.RowNum + 1 & ",""=Allocation Management"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(4, 5).Formula = "=SUMIF( _
             B2:B" & o.RowNum + 1 & ",""=Cash Management"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(5, 5).Formula = "=SUMIF( _
             B2:B" & o.RowNum + 1 & ",""=Purchase Orders"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(7, 5).Formula = "=SUMIF( _
             B2:B" & o.RowNum + 1 & ",""=Student Billing"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(8, 5).Formula = "=SUMIF( _
             B2:B" & o.RowNum + 1 & ",""=Accounts Receivable"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(9, 5).Formula = "=SUMIF( _
             B2:B" & o.RowNum + 1 & ",""=Fixed Assets"",A2:A" & o.RowNum + 1 & ")"
        moWorksheet.Cells(10, 5).Formula = "=SUMIF( _


             B2:B" & o.RowNum + 1 & ",""=Cash Receipts"",A2:A" & o.RowNum + 1 & ")"

        moWorksheet.Range("E1", "E10").NumberFormat = "$#,##0.00_)"

        buildChart 'subroutine that builds a chart based on the worksheet

        'Clean up
        Set moWorksheet = Nothing
        Set moExcel = Nothing

    Else
        ' Fill In The Details
        With moWorksheet

            Dim l As Long
            If o.Field(1) = "Debit" Then
                .Cells(o.RowNum + 1, 1) = o.Field(2)
            Else
                .Cells(o.RowNum + 1, 1) = o.Field(2) * -1
            End If
            .Cells(o.RowNum + 1, 2) = o.Field(3)
        End With
    End If

End Sub

'this sub uses Excel's objects to create a chart through code
Private Sub buildChart()


    Dim oChart As Chart


    moWorksheet.Range("D1:E10").Select

    Set oChart = moExcel.Charts.Add
```

(Continued- page 3 of 3)

```
    oChart.ChartType = xl3DColumnClustered
    oChart.SetSourceData Source:=moExcel.Sheets("Sheet1").Range("D1:E10"), PlotBy:= _
      xlColumns
    oChart.Location Where:=xlLocationAsNewSheet

    With oChart
        .HasTitle = False
        .Axes(xlCategory).HasTitle = False
        .Axes(xlSeries).HasTitle = False
        .Axes(xlValue).HasTitle = False
    End With

    oChart.SeriesCollection(1).Select
    oChart.Walls.Select
    oChart.PlotArea.Select
    oChart.Walls.Select

    With moExcel.Selection.Border
        .ColorIndex = 16
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With

    moExcel.Selection.Fill.TwoColorGradient Style:=1, Variant:=1
    With moExcel.Selection
        .Fill.Visible = True
        .Fill.ForeColor.SchemeColor = 42
        .Fill.BackColor.SchemeColor = 41
    End With

    With oChart.ChartGroups(1)
        .GapWidth = 150
        .VaryByCategories = True
    End With

    With oChart
        .DepthPercent = 100
        .GapDepth = 150
    End With

End Sub
```

Read-Only Database Assistance Samples

Read-Only Database Assistance is an optional module that provides open access to the Blackbaud database structure and its underlying relational architecture.***The Financial Edge*** includes two *Read-Only Database Assistance* sample programs installed in The_Financial_Edge\Help\Samples\Advanced Samples\Open folder. These samples include code to create an auto-refreshing report and an HTML dashboard.

To create a disconnected ADO recordset, use the UtilityCode CreateDisconnectedADORecordset function. This is a read-only recordset you can use to browse through ***Financial Edge*** data while protecting your data from updates and deletes.

If you are using *Read-Only Database Assistance*, pass in your *Read-Only Database Assistance* user name and password. If you are using VBA or API, pass in your vendor name and serial number. Use the sSQL argument to specify the data you want.

Creating an Auto-Refreshing Report

This sample uses *Read-Only Database Assistance* to run queries on ***The Financial Edge*** database so you can easily monitor account and project activity, and it displays the query results in *Excel*. To use the Auto Refreshing Report, browse to the Help/Samples/Advanced_Samples/API/Samples/SamplesinXLS folder and copy both the Samples.xls and Query1.dqy files. Then, paste both files in the Financial Edge/Plug-ins folder. The spreadsheet refreshes automatically every ten minutes, but you can change the refresh interval.

Creating an HTML Dashboard

With the HTML Dashboard sample, you can export the data in ***The Financial Edge*** *Dashboard* to an HTML file you can send to users who do not have ***The Financial Edge*** installed on their computers. This sample uses *Read-Only Database Assistance* to run queries on ***The Financial Edge*** databases.

The HTML Dashboard is located in the Help/Samples/AdvancedSamples/Open/CreateHTMLDashboards folder. To run this sample, you must replace the sample user ID and password with your user ID and password. For more information, see the comments in the code text. When you run the HTML Dashboard, a submenu appears listing the names of all available dashboard panels. To save a panel in HTML, select a panel from the list and click **Save**. After you save the file, a message appears asking if you want to view the file. To view the file in your browser, click **Yes**.

Dashboard panels consisting of only graphs cannot be exported to HTML. You can export panels that contain both graphs and tables, but the graphs will not appear in the HTML file.

Index