iMIS e-Series

iBO Programmer Guide

Version 10.6

Advanced Solutions International, Inc.

Advanced Solutions International, Inc.

Updated on 4/28/2006.

- Sales and Marketing Headquarters
  901 N. Pitt St., Ste. 200
  Alexandria, VA 22314
  800.727.8682
- Center for Technical Excellence
  11500 Metric Blvd., Ste. 150
  Austin, TX 78758
  512.491.0550

Notice

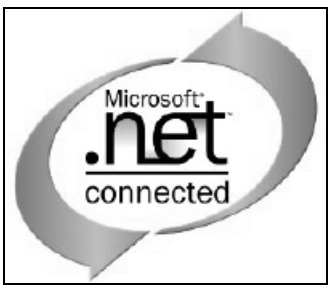
This document and the products it describes are confidential information that is furnished under the terms of a license agreement as Software and associated documentation. They may be used and copied only in accordance with the terms of the license agreement. No part may be copied, transmitted, or reproduced in any form without written permission from Advanced Solutions International, Inc. (ASI).

ASI and the program authors have no liability to the purchaser or any other entity, except as provided in the license agreement permitting your use of this software, with respect to any liability, loss, or damage caused, directly or indirectly, by this software or reliance on this documentation, including but not limited to any interruptions in service, loss of business, anticipatory profits, or consequential damages.

Should this document be marked Draft or describe our plans or products not released for general sale to new customers, by reading it you agree not to use this information for any purpose except to educate you and your organization about the current development intentions of ASI; you agree not to rely upon this information, even for planning purposes, since it can be changed at any time without any notice to you; and you agree not copy, publish, or disseminate the information in this document outside your organization without our written consent.

Trademarks

iMIS is a registered trademark of Advanced Solutions International, Inc. Express!, *iMIS* *e*-Series, *iMIS* LAN, CyberiMIS, *iMIS*.com, *iMIS* for Sybase ASA, *iMIS* for Microsoft SQL, and *iMIS* for MSDE are trademarks, servicemarks, and designmarks of Advanced Solutions International, Inc. Also, *iMIS* *e*-X (where X is the name of the module, such as *e*-Events or *e*-Orders) is a trademark, servicemark, and designmark of Advanced Solutions International, Inc. All companies and products mentioned herein are trademarks or registered trademarks of their respective owners.



Microsoft .NET Connected Logo

The .NET Connected Logo indicates an application or service whose primary functionality is exposed through, or is programmatically enhanced by the consumption of, Web services that comply with industry Web service standards. These applications must also be built on the .NET Framework, a component of the Windows operating system that enables the use of Web services and next generation Windows applications. This certification recognizes solutions that fully support Web service capabilities, and takes advantage of the .NET Framework programming model benefits such as multi-language support, added security, and enhanced flexibility - benefits that customers will also realize. Microsoft and the .NET Logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Designed for Windows XP

Microsoft created the "Designed for Windows XP" logo used on the *iMIS* product to indicate that the product will be stable when running Windows XP, the related software or driver components can be installed or removed easily, and the basic experience with the product and the operating system will be the same or better after upgrading to future versions of Windows. ASI is proud that extensive testing has proved that its products meet all the criteria needed to be able to display this logo. Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Contents

iBO Programmer Guide

Before you begin

iBO purpose and scope

iMIS Business Objects (iBO) includes the core components Contacts, Events, FundRaising, Orders, Subscriptions, and UserSecurity. These COM-based components enable third-party developers to indirectly access the *iMIS* database via class interfaces that closely parallel the functional interfaces of the traditional *iMIS*. These components can be used to build custom web pages that are integrated with the *e*-Series web site.

iBO provides developers with API visibility to the Contacts, Events, FundRaising, Orders, Subscriptions, and UserSecurity components.

- The *Contacts* component provides COM interfaces that enable viewing and modifying contact data, such as the contact's profile, addresses, financials, and user-defined demographic data.
- The *Events* component provides COM interfaces that enable creation and modification of event registrations and reading of event setup data, including functions and pricing structure.
- The *FundRaising* component provides COM interfaces suitable for accepting simple donations.
- The *Orders* component provides COM interfaces for creating new standard orders, adding products to the order, calculating prices, freight, handling, and taxes, and reading existing orders from the database.
- The *Subscriptions* component provides COM interfaces for creating and billing the subscriptions associated with a member type, paying for subscriptions, and querying subscriptions.
- The *UserSecurity* component provides COM interfaces to validate passwords and to determine the ContactID of the logged-on staff person or member.

The iBO library will be expanded and released in phases. New components will follow, while the functionality of existing components will continue to broaden. Initially, the components will match the functionality contained in *e*-Series.

General Availability

iBO is generally available to all who meet these requirements:

- Authorized *iMIS* Service Providers (AiSP)
- Qualified to implement *e*-Series
- Have clients who are licensed for *e*-Customer Management
- Have attended an iBO Implementation training class

Getting support and certification

Contacting ASI Developer Support

ASI offers a variety of support options. These include free services on our website, where you can search our extensive information base and connect with other users of iBO.

Support for iBO is provided on a for-fee basis. Submit issues to ASI Support, and Consulting will work with Support to research the answers you need.

Training and certification

You must become certified in iBO technology to receive your license to implement applications using it. These are the steps to licensing:

1. Complete the first prerequisite: *e*-Series *Implementation* class.
2. Complete the second prerequisite: *e*-Series *Advanced* class.
3. Complete the required iBO training: 2-day technical authorization class.
4. Pass the iBO certification exam.
5. Request an iBO license.

Documentation conventions

The ASI printed documentation for iBO uses the typefaces and symbols described below to indicate special text:

`San-serif`	Indicates sample code and proper names of iBO identifiers, such as classes, interfaces, methods, and variables.
Italics	Indicates code comments, as well as new terms and book titles.
Keycaps	Indicates a keyboard key. For example, "Press **Esc** to exit a menu."
[ ]	In text or syntax listings, encloses the optional or variable items. Do not type the brackets.
< >	In text or syntax listings, indicates a variable string; type in a string appropriate for your code. Do not type the angle brackets. Angle brackets are also used for HTML tags.
…	In syntax listing, indicates code that is missing from the example.

Concepts

Understanding Business Objects

iMIS Business Objects (iBO) are a set of COM components that enable third-party developers to create scalable custom applications that leverage the *iMIS* backend database. iBO components ensure the integrity of the business rules inherent with the existing *iMIS* system.

What are Business Objects?

Mirroring the real world

In the broadest sense, business objects represent your business programmatically. Since organizations run on physical objects and concepts, it simplifies things greatly to model their actual players and processes within self-contained software units that *directly* correspond to real-world entities and activities.

A business object not only contains but also manages its own data. For example, a `Customer` object must include all of the data and routines needed to represent a customer throughout all interactions, so that you can use it unchanged across all the applications for your business. This means that you use the same `Customer` business object in code for taking an order, sending out a bulk mailing, and analyzing sales patterns; no subsequent coding you write can disturb your initial order-processing code: the behaviors are independent and controlled.

Interfaces: The freedom to evolve

Over time, each business object must evolve, gaining properties and methods to support new functionality. However, we try to keep our interface backward compatible so that previous properties and methods (interface) will remain unchanged and your existing applications will work as written without being changed to accommodate the new features.

This freedom from the disruption of future changes allows you to generate rich feature sets at a faster pace by using an iterative, building-block approach - that is, your application is free to evolve as well. Moreover, using business objects containing robust *iMIS* business logic lets you build your own suites of functionality without risking the integrity of the underlying *iMIS* database.

Schema encapsulation

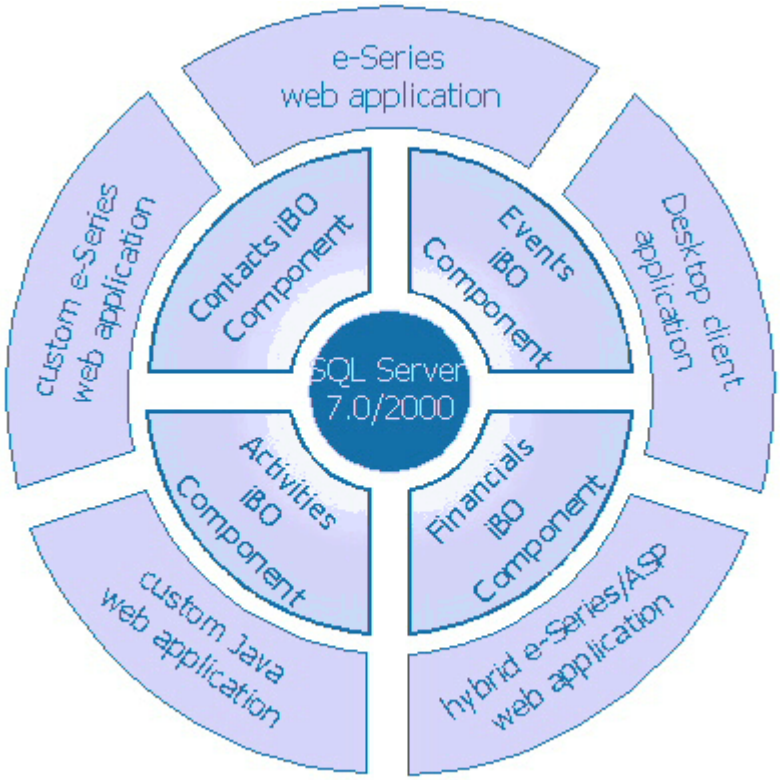
But iBO offers more than interface stability and data validation: because of *schema encapsulation*, you can focus on the specific value of the features you create without having to master the underlying *iMIS* database structure or take on responsibility for the database consistency and integrity resulting from your use of iBO.

Without such encapsulation, a developer of any functionality lying on top of a back-end database must thoroughly understand the database organizational structure (schema). Removing this burden is one of the key benefits to third-party developers.

Inherently, schema encapsulation assures greater *iMIS* database stability because it isolates the effects of executing business rules to just those tables that are directly affected. The resulting clean and modular implementation provides enormous cost savings through reduced complexity and greater scalability.

The general model: Access through layers

The Business Object approach is a generalized model for business objects, intended to encompass many interpretations and implementations. The diagram below illustrates how the *iMIS* business objects form a protective middle layer of a 3-tier architecture, serving as intermediaries between the encapsulated database and the client applications. Other business object systems may implement a superset or a subset of this model.

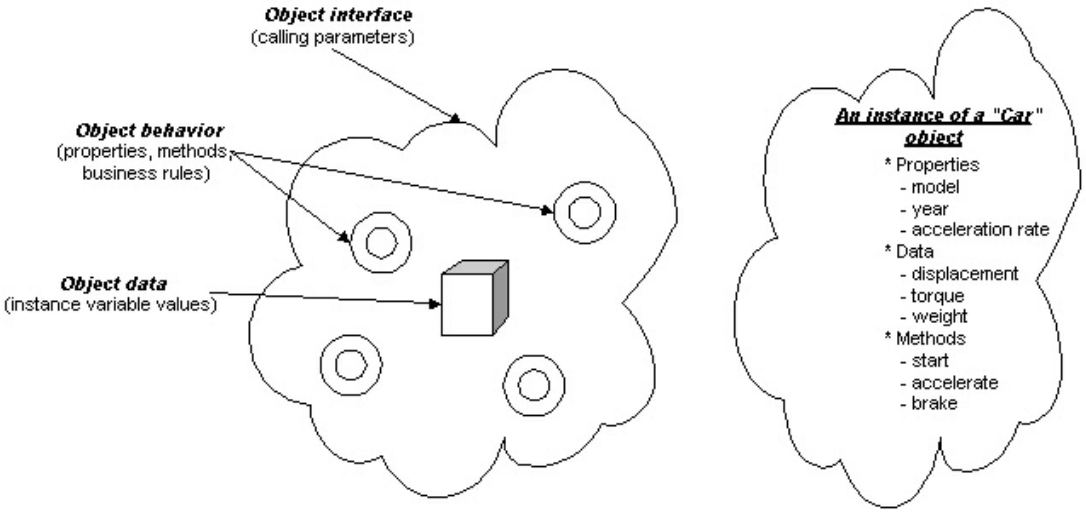


iBO Components as layer between applications and the RDBMS

As shown, the tools underlying the system (such as database systems and technology components) reside, shielded, at the core. Only business objects interface directly with this layer of tools. The business objects are hidden from end-users, who reach them only through visual presentations and desktop programs. These two inner layers, the business objects and inner data core, comprise the foundations for the *iMIS* Business Framework Architecture.

What is inside a Business Object?

Business objects are representations derived directly from the abstract concepts of the object-oriented analysis and design model. A business object consists of data (state information) and behavior (properties and methods) that you access together externally through an interface. The specific objects manipulated by application programs are instances of class definitions, where the class serves as the template from which an object is created. For example, a `Car` class could have an object instance of a BMW, which contains properties describing that particular car (i.e., model, year, acceleration rate) along with the behaviors possible to manipulate it (start, accelerate, brake).



Representation of what business objects contain

Business Object data

The data within an instantiated object is stored within variables that are unique to that specific instance. The values contained within instance variables are particular to each object, even objects of the same class. That way, we can have a `Customer` class, where its instantiated objects have different information for different customers (i.e., *Acme* vs. *Ford*).

These data variables exist as either Public or Private. Public variables are visible to your calling application, which means that you can manipulate their values directly. Private variables are invisible to you, existing only for use by code contained within the object itself. Variables are usually made Private to allow for the enforcing of business or validation rules; rather than set a variable directly, you call a method to do so.

iBO follows this practice of constraining variables to be Private, which carries a major benefit: object internals and variables can change significantly between product releases with little to no impact on applications that use the objects, because the methods retain their original, abstracted interfaces.

Business Object properties

The properties of a business object provide the most powerful means by which a business object designer can choose what information attributes of the object are available to a client program. Properties do not necessarily have a one-to-one relationship with an object's data. For example, the `Car` object diagrammed above may not make its displacement, torque, and weight data variables accessible, yet it does allow clients to avail themselves of the acceleration rate property, as computed from the directly inaccessible data variables.

Properties can be read-only, write-only, or both read and write. They can be set up to grant direct client modification of an object value, or they may have additional validation checks or business rules applied subsequent to submission of the value to be modified.

Business Object methods

Simple access to the property attributes of an object is not enough to model full behavioral interactions of the object with the calling client or other objects. Methods provide the mechanisms for an object to offer much more robust services or functionality. As routines within an object, interrogative methods use their own data or passed-in parameters to return values or other information back to the calling client. Imperative methods merely perform a service as directed by a client, and they may or may not return a status of the function performed.

Who should use iBO?

iBO is intended for programmers working in these situations:

- Association offices
- Authorized *iMIS* Solution Providers (AiSPs)
- Association management firms
- Organizations needing CRM capabilities

In each case, the programmer uses the Contacts component to create a Windows or browser-based interface to an *iMIS* installation.

Note: You must learn how to program with COM-based components before attempting an iBO implementation.

iBO's role in iMIS

iMIS Business Objects are functional components that provide an application programming interface (API) to data within the *iMIS* database. You can incorporate these components in programs written in a variety of languages, such as Macromedia ColdFusion and Microsoft Active Server Pages.

Through the business object's API, incorporating programs gain access to *iMIS* logic. For example, you can use the Contact business object to return member records from the *iMIS* database. By using the *iMIS* business object, all *iMIS* validation and business logic, such as flow-down rules, is securely enforced.

iBO components and classes

Be aware that not all iBO classes and interfaces that are visible to you through object browsers (and IntelliSense) are for public use. Some exist for internal use only.

There are also certain classes and methods which are Use At Your Own Risk. These API items have been built to support other API items; they are lower-level methods. These methods have been implicitly tested by ASI's Quality Assurance Department when they tested the released API methods. However, these methods have not been fully tested by ASI's Quality Assurance Department. You can use these methods with the assurance of binary compatibility for future releases. There is no Technical Support available for these methods.

The way to ensure that you are using iBO correctly is to refer to the online *API Reference* (p. 19), which is an HTML Help (.CHM) file. If a specific method or property is for ASI use only, the reference would make that clear; if an entire class is for internal use, it would not appear in this reference at all.

Functionality overview

iMIS Business Objects offer a versatile tool for extending the functionality of *e*-Series to meet specific client requirements. For example, you could implement a tailored workflow as part of the client's event registration process; following the tailored workflow, you can add the resulting registration to the *e*-Series shopping basket or save it to the *iMIS* database, depending on your client's needs.

The Contact component lets you select an individual contact within the *iMIS* database, provide access to the contact's data, and load all associated address, notes, and activity records. The component lets you create a list of `Contact` records meeting criteria that you specify. You can also create and delete individual `Contact` records through the interface.

The Event component lets you select one or more events, select a particular function, or list the functions within an event. Through the Event component, you can access event and function properties, including pricing information. The component lets you select a specific event registration or a list of registrations, and it lets you create a registration, including payment and badge information.

Invoking server objects

At least two iBO Platform objects must be instantiated before you can instantiate a functional BO object:

1. `iboUserSecurity.CUser` specifies the database to be accessed, and the `CUser` instance is passed to `BOAdmin` objects to establish iBO connectivity. The critical properties in this object are:
   - `DBID` - (*required*) must be an ODBC DSN, formatted as ODBC:DSN_Name.
   - `UserID` - used by `CContact` for change logging. You can call `CUser.Validate` to verify a userID and password. This should be the user logon.
   - `Password` - should be from the logon.
2. `BOAdmin` is the core platform object, which exists specifically for each component:
   - `iboContactManagement.CContactsBOAdmin`
   - `iboEvents.CEventsBOAdmin`

That is, before you can do anything with the `Contacts` component, you must instantiate the `CContactsBOAdmin`.

The only properties that these `BOAdmin` objects have are the `ErrorsCount` and `Errors` object. However, the `BOAdmin` has several methods. You will use these methods to add new data and find existing data.

Naming conventions

While browsing through the API reference, you will see that certain object names are plural. Throughout iBO, a plural name always signifies a collection of objects (*for example*, `Errors` is a collection of `Error` objects).

Plurals are also used for the names of `BOAdmin` methods. For example, `NewRegistration()` creates a new `Registration`, whereas `NewRegistrations()` creates an empty collection of `Registration` objects. You must use the collection object to access the methods for populating the collection.

Note: Each collection object has a property that is its filter: `CBatches` has `BatchFilter`, `CContacts` has `ContactFilter`, and so on. Thus, you can deduce the filter name from that of the collection.

Most collection objects have a similar method call for populating and retrieving objects from them:

- `GetContactById` (singular) retrieves one Contact record from the database, based upon the ID parameter. This method does not populate the collection; rather, it returns the single object (if it exists in the database).
- `GetContacts` (plural) retrieves multiple Contacts. You can pass a `p_lngMaxRecords` parameter to restrict the amount of data returned from the filter, up to a maximum of 500 (that is, if you specify 10 and the filter finds 50, only 10 would be returned). The default returns all of the data specified by the filter.

- `GetContactsPaged` (plural) is identical to GetContacts in that it uses the filter to populate the collection, but this method will restrict the size of the collection. It also returns a reference so that this method can be called a second time and retrieve the next set of data. This paging works much like a web search engine, which displays only a subset of the results per page but provides links to get the next subset, and to the next. Keep requesting pages and testing the number returned; when the number is 0, there is no more data.
- `GetContact` (singular) retrieves one object from the collection, which must be populated first.

There is no parameter to pass an SQL WHERE clause to the `GetContacts` method. This would not only iBO restrict to certain flavors of SQL, but would also make it dependant upon table definitions and their relationships. Therefore, the collection is built upon the filter, which allows iBO to evolve and improve. Because of the abstraction of the filter, not only can table column names change without any recoding of the scripts that use iBO, but the column could be moved into another table, allowing for more normalization and optimization of the database schema.

Interface for Activity

These are the public classes in the Activity component. These classes should be used to read the database and should not be used to write to the database, as there is no validation on write operations. These classes are intended to be used by other *i*BO components.

Class	Used to…
CActivities	Retrieve one or more CActivity instances for viewing
CActivity	View one Activity record
CActivityBOAdmin	Used to instantiate CActivities, CActivity

Interface for Contacts

These are the public classes in the Contacts component:

Class	Used to…
CAddress	View or edit a contact's address entries
CContact	View or edit one Contact record
CContacts	Retrieve one or more Contact records for viewing or editing
CContactsBOAdmin	Used to instantiate CContacts, CContact, and CContMgmtConfig
CContMgmtConfig	Used to access system configuration values
CExtView, CExtField	Access the ExtView (Extended View) objects, which are the equivalent of *iMIS* Customizer Windows data *Notes*: ▪ ExtView names reference windows (not tables), while ExtField names reference columns. ▪ ExtField names are case-sensitive, even if the database instance is not.
CFinancialProfile	View or edit a contact's Name_Fin data
CNote	View or edit a contact's note entries

Interface for DataServer

These are the public classes in the DataServer component:

Class	Used to…
CFilter	Used to set the equivalent of an SQL WHERE or ORDER BY clause for retrieval from the database by a collection class

Interface for Errors

These are the public classes for the Errors component:

Class	Used to…
CError	iBO creates a CError instance when an error condition occurs
Cerrors	Collection of errors

Interface for e-Series

These are the public classes in the *e*-Series component:

Class	Used to…
CESeriesBOAdmin	Used to instantiate CESeriesIntegration and to Restore sessions state after a page turn
CESeriesIntegration	Used for ESeriesIntegration for subscriptions and standard orders, to convert an anonymous user to a member, and to serialize session state into an XML string

Interface for Events

These are the public classes in the Events component:

Class	Used to…
CBadge	View or set badge information
CEvent	View an event definition, including the function details
CEvents	Retrieve and access a filtered set of Event objects
CEventsBOAdmin	Retrieve an instance of an Events, Registrations, or Registration object
CEventsConfig	View system setup information for Events
CFunction	View a function definition, including fees details
CFunctionFee	View a fee definition
CRegistration	Create a new registration or view an existing registration
CRegistrations	Retrieve and access a filtered set of Registration objects
CRegLineItem	View or add a registered function (line item) for an event

Interface for Financials

These are the public classes in the Financials component:

Class	Used to…
CBatch	Used to view Batch records
Cbatches	Used to retrieve Batch records from the database

CFinancialEntities	Used to read Ord_Control rows into CFinancialEntity instances
CFinancialEntity	Used to view Org_Control data
CFinancialsBOAdmin	Used to instantiate CFinancialEntities, CFinancialsConfig, and CBatch
CFinancialsConfig	Used to view System_Params data

Interface for FundRaising

These are the public classes in the FundRaising component:

Class	Used to…
CFRDataMgr	Use to instantiate CGift
CGift	Use to make a simple donation

Interface for Orders

These are the public classes in the Orders component.

Class	Used to…
CKitItemOrderLine	View order line information that is part of a kit product
COrder	View or set order information pertaining to Event registration orders, including the Order Lines
COrderLine	View or set order line information pertaining to Event registration orders
COrders	Retrieve and access a filtered set of Order objects
COrdersBOAdmin	Retrieve an instance of an Orders, Order, Standard Orders, Standard Order, Kit Item Order Line, or Standard Order Line
COrdersConfig	View system setup information for Orders
CStdOrder	View or set Standard Order information, including the Standard Order Lines
CStdOrderLine	View or set Standard Order Line information, including Kit Item Order Lines
CStdOrders	Retrieve and access a filtered set of Standard Order objects

Interface for Products

These are the public classes in the Products component:

Class	Used to…
CBaseProduct	Provide a base location for properties common to many types of product objects
CDuesProduct	View Dues/Subscription product information
CDuesProducts	Retrieve and access a filtered list of Dues/Subscription products
CKit	Retrieve a set of Standard Products designated as a kit (sold as a single Standard Product)
CKitItem	View Standard Product information for a product that is a member of a kit
CProductBOAdmin	Retrieve an instance of a DuesProducts, Product Categories, or Standard Products object
CProductCategories	Retrieve a filtered list of Product Categories

CProductCategory	View Product Category information
CPublicationInfo	View Publication information for Standard Products marked as publications
CStdProduct	View Standard Product information, including Kit and Publication information
CStdProducts	Retrieve a filtered list of Standard Products

Interface for Subscriptions

These are the public classes in the Subscriptions component:

Class	Used to…
CSubscription	View or edit Subscription information
CSubscriptions	Retrieve a filtered list of Subscription objects. When used in conjunction with the Contact object, this class may also be used to create, bill, and pay new Subscription objects.
CSubscriptionsBOAdmin	Retrieve an instance of the Subscriptions or Subscription object
CSubscriptionsConfig	View system setup information for Subscriptions

Interface for SystemConfig

These are the public classes in the SystemConfig component:

Class	Used to…
CCashAccount	Used to view data on CashAccounts records
CCountry	Used to view data on Country_Name records
CMemberType	Used to view data on Member_Type records
CSubscriptionDef	Used to view data on the dues subscription products for a member type
CSysCfgBOAdmin	Used to instantiate CSystemConfig
CSystemConfig	Used to view System_Params and Gen_Tables, and to retrieve CCashAccount, CCountry, CMemberType, CSystemParams, and CTaxAuthorities
CSystemParams	Used to view data on System_Params records
CTaxAuthorities	Collection of related CTaxAuthority instances
CTaxAuthority	Used to view data on Products records with PROD_TYPE = 'TAX'

Interface for UserSecurity

These are the public classes in the UserSecurity component:

Class	Used to…
CUser	Contains data on the user who is logged on

Development Guide

Installing iBO

Requirements

Supported Environments

This release supports the following environments:

- *iMIS* database version 10.5 or greater
- Microsoft ActiveX® Data Objects (ADO) transactions
- Microsoft SQL Server (not Sybase), versions 2000 and 7.0
- Microsoft Windows 2000 SP2, Windows XP
- Microsoft Internet Information Services (IIS) 5.x and COM+

Supported Languages

The *iMIS* Business Objects work with many languages, but they are validated for use and supported only with these:

- Macromedia ColdFusion
- Microsoft Visual Basic Script (VBScript)
- JavaScript (from Netscape and Sun Microsystems)
- Microsoft JScript
- Microsoft Visual Basic
- Microsoft Active Server Pages (ASP)

Firewall Configuration

OLEDB uses port 1433 for TCP/IP. SQL Server listens on port 1433 for default instances of SQL Server. Named instances of SQL Server 2000 dynamically assign a port number.

Use the Server Network Utility to configure a specific port for SQL 2000 to listen on, and then set the firewall to allow that port.

Microsoft's knowledge base article *287932* (http://support.microsoft.com/default.aspx?scid=kb;en-us;q287932) describes SQL Server and firewall configuration.

Remote Database Access

To access your database remotely (over the Internet) with iBO, use a Virtual Private Network. Ensure that the DSName referencing the SQL Server uses the server's name and not its IP address.

Installing iBO on the Client

1. Contact ASI to obtain a license key for iBO.
2. Upgrade the *iMIS* license keys for iBO.
3. Upgrade your database to the latest version of *iMIS*.
4. Remove prior versions of iBO before running the *iMIS* installer: **Start | Control Panels, Add or Remove Programs.**
5. Insert the *iMIS* Server CD and run **Setup.exe**.
6. Select **Install Products.**

7 Select **Server**.

8 Select **Install iBO**.

9 Select **Next** when the **InstallShield Wizard** window displays.

10 Select **Next** to install to the default directory or **Browse** to select another.

Note: Do not rename this directory or any subdirectories after installation, or future upgrades will not run properly.

11 Select **Next**.

12 After *i*BO is installed, select **Finish**.

13 In *iMIS*, update the SQL logins: Select **File> System Setup**, select **User Passwords**, and select **Update All SQL Logins**.

Note: If you do not update SQL logins before running a custom iBO application, you may receive error 3261 or an error indicating that the database cannot be accessed; run the update to correct this.

iBO Installed Files

These files are included an *i*BO installation (directory paths are relative to the main *i*BO installation directory):

Path	Filename	Purpose
<root>	readme.txt	Release info; finding documentation
\bin	iboAccounting.dll	Component handling Accounting
	bioactivity.dll	Component handling Activities
	iboBOInterfaces.dll	Component handling BOInterfaces
	iboContactManagement.dll	Component handling Contacts
	iboDataServer.dll	Component handling database access
	iboErrors.dll	Component handling errors
	iboESeries.dll	Component handling *e*-Series
	iboEvents.dll	Component handling Events/Registrations
	iboFinancials.dll	Component handling Financials
	iboFundRaising.dll	Component handling Fund Raising
	iboOrders.dll	Component handling Orders
	iboProducts.dll	Component handling Products
	iboSubscriptions.dll	Component handling Subscriptions
	iboSystemConfig.dll	Component handling system configuration
	iboTranslate.dll	Component handling translation
	iboUserSecurity.dll	Component handling user security
	iboXMLUtilities.dll	Component handling XML utilities
\Samples	\EventsDemo	Sample implementation, in ASP
	\FundRaisingDemo	Sample implementations, in ASP, VB, and ColdFusion
	\OrdersDemo	Sample implementation in ColdFusion
	\VBTestClient	Sample Visual Basic test client
\Source	iboEnums_ColdFusion.cfm	ColdFusion code defining constants used to specify filters when retrieving data

Path	Filename	Purpose
	iboEnums_Javascript.asp	JavaScript code defining constants used to specify filters when retrieving data
	iboEnums_VBScript.asp	VBScript code defining constants used to specify filters when retrieving data

Configuring iMIS System Setup

As a rule, system options that are set in the *iMIS* environment for iBO are also enforced by the iBO system.

This table lists the setup options, with explanations of how iBO is affected.

Setup Area	Option	Relationship to *i*BO Processing
ID Assignment	Auto Assign Member Numbers	If enabled, *i*BO uses the same counter to generate new ID values for *i*BO Contacts and returns an error if an ID is passed into the new contact
Standard Fields	Institute Type, Major Key Prompt	Available as CContMgmtConfig interface properties; Institute Type is the 'InstitutePrompt' property
	Preferred Sort Order	Not applicable
	Must be Unique	Applicable when a CContact record is entered or modified via *i*BO
	Flow from ID	If enabled, when ID is set, MajorKey is automatically set to the same value
	Prefixes in Full Name	Any prefixes in this list are used in constructing the FullName field value upon saving.
AR/Cash	Multiple Entities	If enabled, multiple entities are supported, including multiple entities at the product level
Chapters	Chapter Prompt	Available as a CContMgmtConfig interface property
	Group[x] prompts	Not applicable
Customers	Home Phone Flow from/to Phone Number at Address	Specifies the mapping for Home Phone
	Work Phone Flow from/to Phone Number at Address	Specifies the mapping for Work Phone
	Fax Number Flow from/to Fax Number at Address	Specifies the mapping for Fax
	E-mail Flow from/to E-mail at Address	Specifies the mapping for E-mail
Options	Use Bill Categories	Visible via the CContMgmtConfig interface as a property of the CContact object, regardless of this setting.
	Use Functional Title	Visible via the CContMgmtConfig interface as a property of the CContact object, regardless of this setting.
	Force Source Coding	Visible via the CContMgmtConfig interface; no impact otherwise.
	Use Birth Date	Visible via the CContMgmtConfig interface as a property of the CContact object, regardless of this setting.

Setup Area	Option	Relationship to *i*BO Processing
	Prompt for Status Date	Not applicable The value is available via the CContMgmtConfig interface.
	Allow Delete from Palette	Not applicable
	Require Gender	Setting to True causes validation to fail if the CContact.Gender property is not set.
Address Usage	Main, 2nd, and 3rd address prompts	Available via the CContMgmtConfig interface.
	Use Company	Not applicable
	Use Title	Not applicable
	Use to Update Chapter	Not applicable
International Prompts		These prompts will be available via the CContMgmtConfig interface.
Zip Options		Not applicable
Address Codes		These settings will be available via the CContMgmtConfig interface.
Note Prompts		These prompts will be available via the CContMgmtConfig interface.
Accumail Options		Not applicable
Indexes		These fields are added to the Search Types on the Member Find window. If a CContact is inserted/updated, the appropriate index table must be updated to enable this lookup. *i*BO updates these as needed.
Change Logging		Not applicable *i*BO automatically logs changes to any CContact property as if they were listed in the Change Logging fields list.
Advanced	Disable Auto Flow down of company address information	Toggles whether address information flows down automatically upon CContact edit
	Use Parent Company Pricing	Does not affect the Contacts component, but affects the pricing of registrations in *i*BO Events
Access Keys		No access keys are used by *i*BO in this release.
Events	Warning For Max Registrants	If set to True, causes a warning to be returned if a registration is created for an Event that is full.

API documentation

Accessing API Help

For API reference documentation, use the HTML Help file, `iBO_API.chm`.

Finding the latest documentation

The latest version is always available on the ASI website, in the HelpNet community. The HelpNet is restricted to licensed users, so you need to sign up to receive a password; go to the Customer Home page to do so: *ASI's Customer Home page* (http://www.advsol.com/template.cfm?section=Customers)

Current documentation and resources reside in the **Document Archives** of the HelpNet Community (**Release Materials** and **iBO** folders) on ASI's website:

Release news, installation notes, resolved issues	*iMIS Release Notes* and related documentation, PDF
Concepts, usage, references, samples	*iBO Programmer Guide*, PDF
Complete API Reference	`iBO_API.chm`, compiled HTML Help
Current Known Issues (as tracked in our database)	Log in to the Support Web page, select Search, specify Area: **Known Issues**

iBO filters

The `CFilter` object lets you specify the rows to be returned or modified. The `CFilter Translate` method returns conditions that you can place in a `WHERE` clause. It generates join conditions automatically, so you do not need to include them in the `CFilter`.

Types of filters

There are two types of filters: atomic and compound.

- *Atomic* filters specify a single condition, such as `ZCContacts_CContact.County, "TRAVIS", efilEqual`
- *Compound* filters have two or more atomic filters joined by a Boolean operator (AND , OR, etc.).

How filters work

Outside of a component, `CFilters` are defined in terms of properties in the encapsulating component's interface, rather than in terms of columns in a table. This approach helps to preserve schema encapsulation.

A component can select rows within its encapsulated tables using joins on tables that it does not encapsulate.

For a compound `CFilter`, AND is the assumed Operator if none is specified. If you call the `AddFilter` method on a `CFilter` that already has its properties set, the following happens:

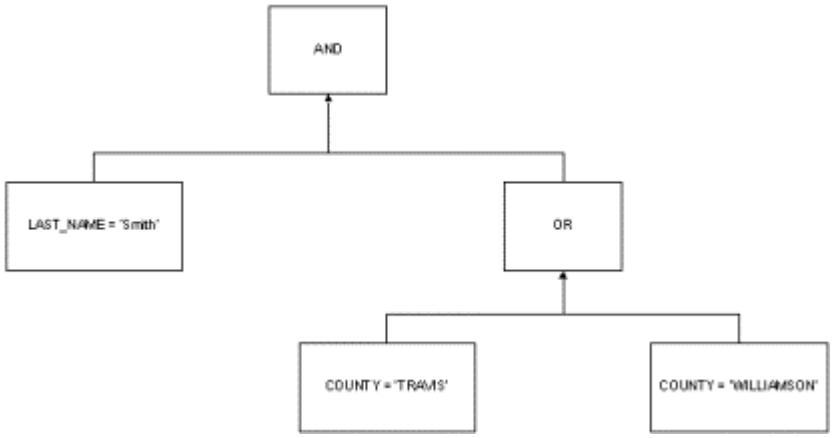
- The `CFilter` is converted to a compound AND `CFilter`.
- The existing filter is added to the new compound filter.
- A new `CFilter` instance is created using the passed values.
- The new instance is added to the compound filter.

Filter example

The following example shows how a filter is created:

```
WHERE (Name.LAST_NAME = 'Smith') AND ((Name.COUNTY = 'TRAVIS') OR
(Name.COUNTY = 'WILLIAMSON'))
```

The best way to model the Where clause is using a hierarchical tree, with the Operator representing a compound filter and the SQL condition representing the atomic filter:



Visualizing the structure of a filter

You would write the following calls to create this filter:

```
Set filFilter = New CFilter

' make the filter a compound AND filter
filFilter.Operator = efilAnd

' create an atomic filter in the compound AND filter
Call filFilter.AddFilter (ZCContacts_CContact.LastName, "Smith”,
efilEqual)

' create a compound OR filter within the compound AND filter
Set filCounty = filFilter.AppendFilter(efilOr)

' create the two atomic filters within the compound OR filter
Call filCounty.AddFilter(ZCContacts_CContact.County, "TRAVIS",
efilEqual)
Call filCounty.AddFilter(ZCContacts_CContact.County, "WILLIAMSON",
efilEqual)
```

When to add joins

The DataServer uses inner joins when columns from more than one table are involved in a DataRequest. When the tables involved in a DataRequest are all encapsulated by the same component, the join is *IntraComponent*; when the tables are encapsulated by two or more components, the join is *InterComponent*.

The DataServer handles IntraComponent joins without help. It can create inner joins between all of the tables encapsulated by a component. For such joins, there is no special processing required: Use `CFilter.AddJoin` to specify which rows to select.

However, the DataServer does need help with InterComponent joins, which involve multiple components. You use the `CFilter.AddJoin` method to tell the DataServer which properties in each component to use to build the inner join. The DataServer associates one or more inner join clauses with each combination of two properties. The following InterComponent joins are supported:

- `iboContactManagement.CContact.ContactID = iboOrders.COrder.ShipToID`
- `iboContactManagement.CContact.ContactID = iboEvents.CRegistration.RegistrantID`
- `iboOrders.COrderLine.ProductCode = iboEvents.CFunction.FunctionCode`
- `iboOrders.COrder.OrderNum = iboEvents.CRegistration.OrderNum`

These InterComponent joins let you relate iboContactManagement, iboEvents, and iboOrders. The DataServer requires the help of the `CFilter.AddJoin` because there are several possible ways to relate iboOrders to iboEvents.

Example: Adding a join

```
Sub addJoinTest()
    Dim mystring As String
    Dim i As Long
' Example: AddJoin filters -- finds all contacts registered for an
event.

' All events with EventCode <> 0
m_objContacts.ContactFilter.AddFilter
ZiboEvents_CRegistration.EventCode, _
”0", efilNotEqual
    If handleError(m_objContacts) Then Exit Sub

    ' All contacts with a contactId > 0
m_objContacts.ContactFilter.AddFilter
ZiboContactManagement_CContact.contactID, _
"0", efilGT
    If handleError(m_objContacts) Then Exit Sub

' Join the Contacts object and the Registrations object on the ContactID
key
m_objContacts.ContactFilter.AddJoin _
ZiboContactManagement_CContact.contactID, _
ZiboEvents_CRegistration.registrantID
    If handleError(m_objContacts) Then Exit Sub

    ' Get all matching contacts
    m_objContacts.GetContacts True, 0
    If handleError(m_objContacts) Then Exit Sub
    For i = 1 To m_objContacts.count
        Set m_objContact = m_objContacts.GetContact(i)
        If handleError(m_objContacts) Then Exit Sub
mystring = mystring & m_objContact.contactID & ";"
    Next i

    MsgBox "count=" & m_objContacts.count & vbCrLf & mystring
End Sub
```

Enumerator constants for filters

The `\Source` folder contains several different scripting language files that define the constants you will use to specify filters when retrieving data:

- iboEnums_ColdFusion.cfm (ColdFusion code)
- iboEnums_Javascript.asp (Javascript code)
- iboEnums_VBScript.asp (VBScript code)

These scripting language files are located in the **Source** folder in the iBO installation folder. The *e*-Series installation process automatically copies the ColdFusion scripting language files to the correct folder (the **Customer/Source** folder used by *e*-Series) for use with *e*-Series. The scripting language files should be copied to the Web Application folder for ColdFusion applications that do not rely on *e*-Series and for non-ColdFusion applications.

You should include the following line of code in every iBO page or in the *Application*.**cfm** file for the application:

```
<CFInclude \ template="../Source/iboEnums_ColdFusion.cfm">
```

The following table lists some of the iBO enumerator constants for component filters. For a listing of all iBO enumerator constants, you can refer to the iboEnums_ColdFusion.cfm, iboEnums_Javascript.asp, or iboEnums_VBScript.asp files, look up the table, and then find the column under the table.

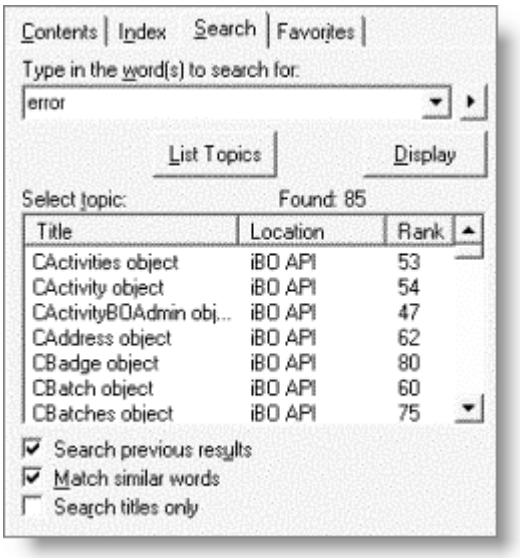
Use with iboDataServer.CFilter.Comparison	Const efilEqual
	Const efilNotEqual
	Const efilGT
	Const efilLT
	Const efilContains
	Const efilBeginsWith
Use with iboDataServer.CFilter.Operator	Const efilAnd
	Const efilOr
	Const efilAtomic

Finding Help topics

The HTML Help offers three main ways to locate information:

1. Using the **Contents** tab, browse the hierarchical structure of the help topics, clicking book icons to expand and collapse subtopics. This is most helpful when you want to explore the information from the top down.
2. Using the **Index** tab, enter a keyword, select **Display**, and browse the topic titles that matched the keyword you specified. This is a fast way to find the primary topics for a known item, such as a particular method.

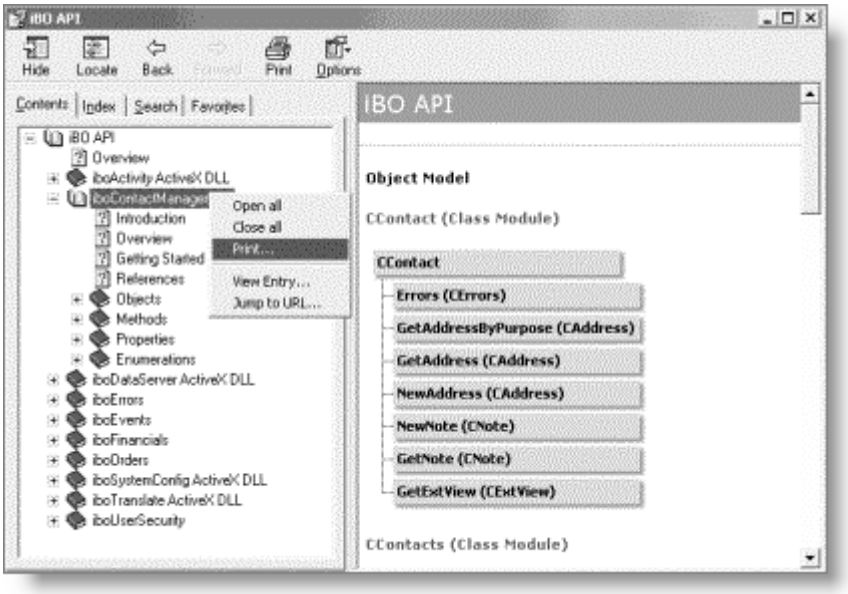
   Tip: Having opened your topic, you can select the **Locate** button on the toolbar to open the **Contents** to your topic's hierarchical location.
3. Using the **Search** tab, enter a search string, select **List Topics**, and browse the results that are returned. By clicking the table headings (**Title**, **Location**, **Rank**), you can sort the results to help you identify likely candidates. If you get too many results, you can do a subsearch by enabling the **Search previous results** checkbox or restrict searching to **Search titles only**:



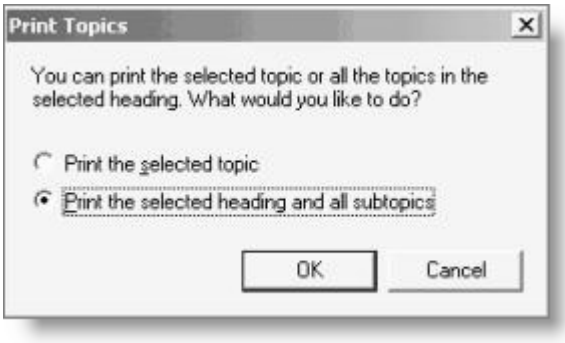
When you find a topic of interest, double-select it or select **Display**.

Printing topics

To print the complete documentation on a given component or object, select its folder (book icon) in the **Contents** pane and select **Print** on either the toolbar or the context menu:



Specify that you want to print the heading and its subtopics:



Development environments and deliverables

ColdFusion versus ASP scripting

ColdFusion supports the use of COM objects, including iBO. Because ColdFusion is the webserver technology underlying *e*-Series, it simplifies your environment to use ColdFusion for your custom *e*-Series pages.

ASP (Active Server Pages) is Microsoft's webserver scripting language. When considering ASP, be aware that you might find it harder to integrate custom *e*-Series pages with ASP than with ColdFusion.

Note: To implement ASP code, you must make minor changes to your *e*-Series source code. Contact ASI Consulting if you are interested in attempting an ASP implementation.

These are examples of how you will simplify your tasks by choosing to script in ColdFusion:

- You can keep the top, side, and bottom indexes synchronized with the rest of *e*-Series with no additional effort if the scripting language you use is ColdFusion.

- If you are going to save meeting registrations to the *e*-Series checkout basket, you must run a ColdFusion script to retrieve certain ColdFusion client-processing variables that are required by the *e*-Series checkout basket.

Note: With any scripting technology, you can edit the files with Notepad or any editor that can save plain text files.

Prototyping in Visual Basic

The fundamental problem with developing applications using current webserver technology is the lack of a robust debugger. Visual Basic version 6 has good debugger, and it has an object browser that presents all of COM components in a summary form.

There are many practical benefits to prototyping your applications in Visual Basic and using its debugger:

- You can easily identify all of each iBO object's methods and properties, including their current values.
- You can change an object's properties.
- You can drill down to child object methods and views.
- You can re-execute code after errors are encountered.

These features can greatly reduce your coding time, so much so that it may take you less time to create a prototype VB application that mirrors the web process exactly (in terms of iBO use) than it would take you to then port it over to the webserver technology.

Avoid creating ActiveX controls

It is technically possible to create ActiveX controls using iBO. Nevertheless, ASI does not recommend the use of ActiveX controls for widespread use on a website, due to security issues and other complications. However, you may find ActiveX controls beneficial for use in an intranet environment.

Unsupported formats: Xtenders, VB.NET

ASI will not support Xtender applications that use iBO.

ASI has not tested iBO with VB.NET and therefore cannot support custom iBO *e*-Series solutions that use it.

Managing the state of web sessions

By convention, iBO provides two methods on all business objects: `GetXML()` and `LoadXML()`. You can use these methods in your client web application for session state management, if desired.

- `GetXML()` returns an XML string, which comprises a snapshot of the object at a point in time (including any modifications of properties, etc., that have occurred).

  If the client application stores this string along with a session ID, it may be retrieved on a subsequent page, a blank object instantiated (by calling `New*() - NewContact()` for instance, with the initialize parameter set to `False`), and a call made to `LoadXML()`, passing in the XML string.

  To avoid getting unpredictable results when calling `LoadXML()`, do not modify the string returned from `GetXML()`.

- `LoadXML()` populates the object to the state it was in when the call was made to `GetXML()` on the previous page.

Each class interface has two error properties:

- ErrorsCount
- Errors

You can call `ErrorsCount` at any time to check for error conditions. `Errors` returns an instance of the `iboErrors` object.

Processing Errors

```
Dim errErrors  as CErrors
Dim errError as CError
Dim i as long
```

Step 1: Get the primary (last) error message:

```
Call objBO.Method
If not objBO.ErrorsCount = 0 then
    Set errErrors = objBO.Errors
    Msgbox "Error description " & errErrors.GetErrorMessage
    objBO.ClearErrors
End if
```

After dealing with an error, clear the `CErrors` instance.

Step 2: Iterate through all error messages:

```
Call objBO.Method
If not objBO.ErrorsCount = 0 then
    Set errErrors = objBO.Errors
    For i = 1 to errErrors.Count
        Set errError = errErrors.GetError(i)
        Msgbox "Error number " & errError.Number & ", " &
_errError.Message & ", " & errError.Category & ", " & errError.Location
    Next i
    objBO.ClearErrors
End if
```

After dealing with an error, clear the `CErrors` instance.

`objBO` will return without processing whenever `objBO.ErrorsCount` is not zero.

Using a General Error Handling Procedure

The amount of code you would write to handle errors could exceed that of your iBO-related code itself if you do not use a general procedure to handle the errors. We use “general” to suggest reusability.

There are many ways of exposing a general procedure to handle iBO errors. Several ways are listed above. Which method is appropriate for you depends on the nature of your project.

To handle errors in Visual Basic

We recommend using VB to prototype applications, and you will find it most efficient to put the generic error handling routine in a VB module (.BAS). You can include the single .BAS file in all iBO-related projects, which drastically cuts down the time required to setup the iBO inline error/status checking. Following is an example of such a module:

```
Option Explicit
Public Function handleError(ByVal p_object As Object) As Boolean
Dim errErrors As iboErrors.CErrors
Dim errError As iboErrors.CError
Dim i As Long
```

```
handleError = True
On Error GoTo ErrorHandler

If p_object Is Nothing Then
printDebugMsg "Could not instantiate iBO object."
Else
If p_object.ErrorsCount > 0 Then
Set errErrors = p_object.Errors
For i = 1 To errErrors.Count
Set errError = errErrors.GetError(i)
printDebugMsg "Error " & i & ": " and errError.Number _
and "; " and errError.Message _
and "; Location: " and errError.Location
Next i
errErrors.ClearErrors
Else
handleError = False
End If
End If
Exit Function

ErrorHandler:
printDebugMsg "exception in iBO Error Handler"
End Function
Private Sub printDebugMsg(p_string As String)
Debug.Print p_string
MsgBox p_string, vbCritical, "iBO Error"
End Sub
```

This function does several things:

- Assumes failure
- Checks to make sure iBO was even instantiated
- Reports every error, not just the first one
- Clears the iBO object's `Errors` object
- Reports runtime errors encountered trying to report the errors
- Reports success only after verified error free

Further, this function can be modified in one place. If a change in the way errors are to be formatted to the user changes, then the code does not have to be pasted all over the rest of the project. Also note that besides displaying the error in a MsgBox, it will also print the error to the VB debugger's 'Immediate' window, so there is an audit trail of the errors generated.

This is how simple the code can become after implementing this .BAS file with this global procedure:

```
On Error GoTo fatal_err:
'This call will create a registration with auto-enroll on
Set reg = cbo.NewRegistration(CUser, _
"ANNUAL", "101", True, True, True, True)
If handleError(cbo) Then
    Exit Sub
End If
'If only auto-enroll and then goto checkout basket are required,
SubmitChanges() save it to basket
reg.SubmitChanges
If handleError(reg) Then
    Exit Sub
End If
Exit Sub
fatal_err:
MsgBox Err.Description
```

Notice that the entire iBO object is passed to the error-handling routine. This alleviates having a thorough knowledge of the error handling by all programmers involved in the iBO projects. The benefit of this is to allow for less technical analysts to prototype the *e*-Series customization, and let the technical web programmer craft the finished solution.

To handle errors in ColdFusion

You can create a global error handling routine in ColdFusion using the `<CFINCLUDE>` tag. However, the error handling when scripting on a webserver cannot be as simple as the VB prototype. In the VB prototype, you simply report the error and stop; in the webserver script, the web page must be completed regardless. Leaving an error message on a half-drawn web page is not comforting to the end user. If the processing is being done in an included script, that script could be aborted, but still the error must be propagated up to the script that is drawing the page.

Despite the additional work required, you have much to gain from using a generalized procedure, such as follows:

```
<CFSET errMessage="">
<CFIF PARAMETEREXISTS(errErrors) EQ "NO">
<CFSET errMessage="Could not instantiate iBO object.">
<CFELSE>
<CFIF errErrors.Count GT 0 >
<CFSET errMessage="">
<CFLOOP INDEX="i" FROM ="1" To = ".errErrors.Count."">
<CFSET errError = errErrors.GetError(i)>
<CFSET errMessage="iBO Error .errError.Number.;
.errError.Message.;
Location: .errError.Location.<BR>">
</CFLOOP>
<CFSET blnResult = errErrors.ClearErrors()>
</CFIF>
</CFIF>
<CFIF errMessage GT "">
<CFPARAM NAME="CLIENT.ID" DEFAULT="">
<CFFILE ACTION="Append"
FILE="C:\e440\iBOErrors.txt"
OUTPUT=".Now(). IP=.CGI.REMOTE_ADDR. iMIS ID=.CLIENT.ID.
.errMessage."
ADDNEWLINE="YES">
</CFIF>
```

Besides looping through all of the errors, putting them in one string, and clearing the `Errors` object, this procedure reports the error to a text file so that the webmaster can review what iBO errors have been generated through live usage. The following are included:

- The IP address, to help identify anonymous users
- The *iMIS* ID (if the user logged into *e*-Series)
- The date/timestamp of the occurrence
- The actual error

The following example shows how to properly invoke this global error procedure in the ColdFusion script:

```
<!---This call will create a registration with auto-enroll on--->
<CFSET reg = cbo.NewRegistration(CUser,"ANNUAL",
".CLIENT.ID.",True,True,True,True)>
<CFSET errErrors = cbo.Errors>
<CFINCLUDE TEMPLATE="iBOHandleErrors.cfm">
<CFIF errMessage GT "">
<P><FONT size="3" color="red"><CFOUTPUT>
.errMessage.
</CFOUTPUT></FONT></P>
```

```
<CFELSE>
<!---If auto-enroll and then goto checkout basket --->
<CFSET blnResult = reg.SubmitChanges()>
<CFSET errErrors = reg.Errors>
<CFINCLUDE TEMPLATE="iBOHandleErrors.cfm">
<CFIF errMessage GT "">
<P><FONT size="3" color="red"><CFOUTPUT>
.errMessage.
</CFOUTPUT></FONT></P>
</CFIF>
</CFIF>
```

The ColdFusion global procedure requires more coding than in VB. The error object cannot be passed as a parameter, but must be 'set' before the global script is 'included'. Though the global script could output the error, it would then output the error for every script. This may or may not be appropriate for every web page. For instance, if the processing was done before the `<HTML>` tag was issued, the error will be wiped out when the real web page starts. If the error occurred when the processing was to draw a number in a narrowly defined table column, the output could be a few characters per line over dozens of lines, which would look strange.

Note: In this example, every iBO method call causes the `IF` statements to nest another level deeper. Although indenting for this will cause your script to scroll far to the right, it is the best way of resolving the end of each `IF` statement.

You must take special care when using the `<CFINCLUDE>` tag, because paths presented in the TEMPLATE are relative to the script that *calls* the include. `/ScriptContent` is the easiest place to put `iBOHandleErrors.cfm` because, if your calling script is also in that folder, you can avoid specifying the path. If your calling script is in the `/Source/Meetings/` folder, then the relative path you would use is `"../../ScriptContent/"`. This relative path is also kept in the ColdFusion variable `CLIENT.CLIENTPATH`.

iBO Error messages

```
Enum EIBOErrNumber
' Maximum resource ID is 65,000
' General Error Numbers
eGenUnknownError
1000
Unknown error. %1
eGenItemNotFound
1001
Item not found. %1
eGenSAXParsingError
1002
Error in XML. %1
eGenDateFormatError
1003
Date is invalid. Please format as MM/DD/YYYY or MM/DD/YYYY hh:mm:ss. %1
eGenInvalidParameterError
1004
Invalid parameter. %1
eGenUnsupportedFunctionality
1005
Function is not supported. %1
eGenGeneralError
1006
General error. %1
eGenItemExists
1007
Item exists. %1
eGenItemDoesNotExist
1008
Item does not exist. %1
```

```
eGenErrorExists
1009
Error exists from previous call; request not processed.

' iboContactManagement error numbers start at 1300
(1000 + (300 * Component Enum)
eContInvalidAddress
1300
Invalid Address entry. %1
eContInvalidContact
1301
Invalid Contact entry. %1

' iboEvents error numbers start at 1600
eEvtsMaxRegistrantsExceeded
1600
Warning: The maximum number of registrants for the event has been
exceeded.
eEvtsDuplicateRegistration
1601
That contact is already registered for the selected event.
eEvtsInvalidFunctionCode
1602
[%1] is not a valid function code for the event.
eEvtsNoPricingForRegClass
1603
No pricing is defined for function: %1, for the selected registrant
class.
eEvtsBatchNumberRequired
1604
Batch control is turned on. A batch number is required for the
registration.
eEvtsNoLineItems
1605
There are no line items for this registration.

' iboActivity error numbers start at 1900

' iboOrders error numbers start at 2200

' iboUserSecurity error numbers start at 2500
eUsrInvalidUser
2500
Invalid User entry. %1
eUsrCantHashPassword
2501
Error hashing password.

' iboFinancials error numbers start at 2800
eFinInvalidCCNumber
2800
Invalid credit card number %1.
eFinInvalidCardName
2801
Credit card holder name is required.
eFinExpiredCard
2802
Credit card has expired.
eFinInvalidCardType
2803
Invalid credit card type %1.

 ' iboErrors error numbers start at 3100
eErrErrorNumberInvalid
3100
Error number %1 is invalid. Calling method is %2.
eErrDescriptionValue
3101
Value of description parameter is
```

```
eErrUnavailable
3102
Unavailable

' iboDataServer will use numbers assigned to iboErrors
' These messages are also used by the methods in the CTranslate Class in
each BO
eDsvReadNoTransaction
3200
Connection Type is "Read" and Commit Type is not "NoTransaction". This
combination is invalid
eDsvWriteNoTransaction
3201
Connection Type is "Write" and Commit Type is "NoTransaction". This
combination is invalid
eDsvConnectionAlreadyOpen
3202
Can't open a connection which is already open."
eDsvConnectionAlreadyClosed
3203
Can't close a connection which is already closed."
eDsvInvalidProperty
3204
Property %2 in interface %1 is invalid.
eDsvInvalidTranslatedColumn
3205
Translated column %1 is invalid for component %2 interface %3 property
%4
eDsvInvalidTable
3206
Table %1 is invalid. Component is %2.
eDsvInvalidRealColumn
3207
Real column %1 is invalid. Virtual column name is %2 in table %3.
eDsvInvalidColumn
3208
Column %1 in Table %2 is invalid.
eDsvInvalidComponent
3209
Component %1 is invalid.
eDsvCantGetTranslateInterface
3210
Column or property %1 is invalid. Parsed to component %2 table/interface
%3 column/property index %4
eDsvInvalidColumnProperty
3211
Property/Column %1 is invalid."
eDsvInvalidFilter
3212
Filter is invalid.
eDsvErrGeneratingSQLStatement
3213
Error generating SQL statement.
eDsvInvalidFilterOperator
3214
Filter operator is invalid; must be EfilOperator.efilAnd or efilOr.
eDsvInvalidInterComponentJoinIndex
3215
Join index %1 on Property %2 and %3 in invalid.
eDsvErrorCombiningJoinArrays
3216
Error occurred while combining JoinArrays.
eDsvFilterNotInitialized
3217
Filter has not been initialized.
eDsvAllColumnsInvalidInFilter
3218
ZAllColumns is not valid for use in a Filter.
```

```
eDsvInvalidFilterComparison
3219
Filter comparison is invalid.
eDsvInvalidInterface
3220
Interface %1 in Component %2 is invalid.
eDsvPropertyNotInComponent
3221
Property %1 is not in Component %2.
eDsvPropertyIndexNotInInterface
3222
Property Index %1 in interface %2 is not in component %3.
eDsvJoinInvalidInComponent
3223
Join of Table %1 to Table %2 in Component %3 is invalid.
eDsvInvalidIntraComponentJoinIndex
3224
IntraComponent Join Index %1 is invalid in Component %2.
eDsvInvalidInterComponentPropertyJoinIndex
3225
InterComponent Property Join Index %1 is invalid in Component %2.
eDsvTranslatedColumnFilterInvalid
3226
Can't translate Column Filter translated from a Property Filter.
eDsvErrorTranslatingVirtualColumn
3227
Error occurred while translating Virtual Column %1 for Real Column %2.
eDsvErrorAddingTableToCollection
3228
Error adding Table %1 to collection.
eDsvErrorTranslatingColumn
3229
Error occurred while translating Column %1.
eDsvInvalidColumnPropertyType
3230
ColumnProperty Type of Column %1 is invalid.
eDsvErrorOpeningConnection
3231
Error opening connection.
eDsvErrorInitializingRecordset
3232
Error initializing recordset.
eDsvErrorClosingConnection
3233
Error closing connection.
eDsvErrorUpdatingTableList
3234
Error updating table list for inter component joins.
eDsvErrorCreatingJoinArray
3235
Error creating intra component join array.
eDsvErrorJoinRequiredForMultipleTables
3236
Error generating FROM SQL clause. Fields have been requested from
multiple tables but joins have not been specified. CFilter.AddJoin must
be called for inter component joins.
eDsvErrorSQLUpdateColumnsNotSet
3237
Error generating SQL Update statement. The SetValue method must be
called for the TimeStamp column and at least one other column.
eDsvSetValueEnumNotColumn
3238
SetValues was called with an enum value of %1 which does not represent a
column.
eDsvZAllColumnsNotValid
3239
SetValues/SortList was passed ZAllColumns as an Enum. This is not valid.
```

```
eDsvVirtualColumnsInvalid
3240
Column %1 is virtual; virtual columns are not allowed in a SQL
Update/Insert statement.
eDsvSetValuesColumnsFromMultipleTables
3241
SetValues can not be passed columns from more than one table.
eDsvInvalidBoolean
3242
Column %1 is Boolean. Value must be "TRUE" or "FALSE"
eDsvFilterRequired
3243
SQL Update/Delete statement requires a Filter which identifies the row
to modify.
eDsvMultipleTablesNotAllowed
3244
SQL Update/Insert statement can't reference more than one table.
eDsvTimeStampRequired
3245
TimeStamp column must be set for Update and Delete SQL statements.
eDsvErrorSQLInsertColumnsNotSet
3246
Error generating SQL Insert statement. The SetValue method must be
called for at least one column.
eDsvTimeStampNotAllowed
3247
The TimeStamp column can't be specified in a SetValue call for a SQL
Insert statement.
eDsvInvalidConnectionTypeForModify
3248
The connection type must be "Write" when updating the database.
eDsvInsertError
3249
Row was not inserted.
eDsvInvalidParmList
3250
Parameter list must contain pairs of Column Enum and
Ascending/Descending Enum.
eDsvADONoCurrentRow
3251
GetValues called when there are no values. ExecSelect may not have
returned any rows, EOF may have been reached via a Move call.
eDsvColumnNotInSelectList
3252
Column %1 requested by GetValues does not appear in Select List; use
SetSelectList to specify the column before calling ExecSelect.
eDsvInvalidJoinArray
3253
Invalid join array; unable to generate SQL FROM clause.
eDsvJoinInvalid
3254
Join of Table %1 to Table %2 is invalid.
eDsvInvalidDataSource
3255
Invalid DataSource %1. DataSource name must start with literal "ODBC:"
eDsvErrorTableNotInJoin
3256
Error generating FROM SQL clause. Table %1 does not participate in a
join. CFilter.AddJoin must be called for inter component joins.
eDsvInvalidMaxRecords
3257
Maximum number of records must be greater than 0.
eDsvNoTablesSpecified
3258
There are no tables specified in the SQL statement. Use
CFilter.AddFilter, SetSelectList, SetValues or SetOrderList to specify a
table.
```

```
eDsvAsiGopherFailure
3259
Unable to access database using ASIGOPHER login for DSN %1.
eDsvLoginPasswordNotSet
3260
The System_Params value for IBO_Control.Access has not been set in
database %1
eDsvAccessSystemParamInvalid
3261
The System_Params value for IBO_Control.Access is invalid in database
%1.
eDsvUnableToAccessDatabase
3262
Database %1 can't be accessed by IBO.
eDsvLicenseStringNotSet
3263
The System_Params value for System_Control.LicensedProducts has not been
set in database %1
eDsvNotLicensed
3264
Database %1 is not licensed for the requested operation.
eDsvWriteConnectionRequired
3265
%1 requires a ConnectionType of eWrite.
eDsvStoredProcedureMissing
3266
The stored procedure %1 is missing from database %2.
eDsvRecordsetNotOpen
3267
Recordset must be open before calling %1.
eDsvConnectionNotInitialized
3268
InitConnection must be called to initialize a connection before any
other methods are called.
eDsvSqlStatementError
3269
Error generating SQL statement %1
eDsvNoColumnsInSelect
3270
One or more columns must be specified using SetSelectList before calling
ExecSelect.
eDsvInvalidSQLDataType
3271
This property can't be used with the efilGT or efilLT operators. This
property translates to a SQL text, ntext, or image data type.
eDsvCantInitIboCrypto
3272
Unable to initialize cryptography component for DSName %1
```

Integration Guide

e-Series Integration

IBOGuest ContactID

All operations prior to CContact.SubmitChanges can be performed as IBOGuest. *e*-Series can be used to put products in the basket, and you can use iBO to add items to CStdOrder, add address and security information, set the member type, and create and bill subscriptions. GetXML and Restore can be called repeatedly as needed to transfer OrderLines to *e*-Series and back to iBO.

SubmitChanges cannot be called on IBOGuest, and you cannot call SaveToESeriesBasket before calling SubmitChanges since every row in the Basket_Dues table must have a corresponding row in the Subscriptions table.

CUser.UserID does not have to be set to "IBOGuest" for an anonymous user. If CUser.UserID is set to "IBOGuest," ConvertGuest changes CUser.ContactID from "IBOGuest" to the new ContactID. This also applies to CStdOrder.BillToID.

CUser.UserID does not log in a user. If CUser.UserID and CUser.Password are set, the user ID and password can be verified. CUser.UserID is only set by iBO when ConvertGuest is called to change "IBOGuest" to the new ContactID.

If ConvertGuest is called on any ContactID other than "IBOGuest," an error message is generated.

If a custom iBO page returns to *e*-Series checkout without calling the *e*-Series login page, *e*-Series requests that the user register and sets the user up as a Web Member. CESeriesIntegration.ConvertGuest must be called, followed by CContact.SubmitChanges, then the *e*-Series login page. The *e*-Series login page verifies that the user is valid by using the Name_Security row that was inserted by SubmitChanges. An "IBOGuest" CContact cannot be used for SubmitChanges.

A CContact with an ID of IBOGuest cannot be created. If GetContactByID("IBOGuest") is used, you cannot call SubmitChanges on it. You can perform any other operations, but you will not be able to change the database. Because it has different initialization values, it may behave slightly differently than a "normal" new CContact.

Two "IBOGuest" instances cannot interact. Multiple "IIBOGuest" instances have the same ContactID, but are different in memory.

ColdFusion sessions

A ColdFusion session is identified by CFID and CFToken. A new ColdFusion session is assigned a CFID and CFToken when the session is established. The CFID and CFToken are valid until the session times out due to user inactivity. iBO uses CFID and CFToken to identify the Basket table rows associated with the CESeriesIntegration instance.

If invalid CFID and CFToken values are passed to iBO, iBO does not know that these values are invalid. iBO uses these values to read the Basket tables and will not find any rows, which implies that no products have been ordered and no payments have been made for subscriptions.

ESeriesIntegration.GetStdOrder gets the CStdOrder associated with the session specified by CFID and CFToken. iBO then creates a new standard order and adds order lines for each row in the Basket_Order table. This behavior does not correspond to CStdOrders.New or GetStdOrder.

A reference to CContact should not be obtained before calling CESeriesIntegration. If a CContact and its subscriptions are obtained, the SaveToESeriesBasket method is not available since a CContact cannot be passed to CESeriesIntegration. Therefore, you cannot set the CSubscription.Amount paid and process it with *e*-Series checkout. You can modify the subscriptions and then call SubmitChanges to update the Subscriptions table so that *e*-Series can then process these subscriptions in the same manner it uses to process existing subscriptions for existing members. In this scenario, the ConvertGuest method is not available to convert CContact from an anonymous user to a member.

If you call GetXML, save the contents in a ColdFusion session variable, build an HTML string and return it to CF/IIS, and then exit your ColdFusion/custom iBO page, the HTML string displays in the user's browser. If the user clicks a link in the page, an HTTP Request string is sent to IIS. This string is transferred to ColdFusion because of the page name extension. ColdFusion parses the HTTP Request string and extracts CFID and CFToken. The CFID and CFToken values are used to restore the ColdFusion environment. If control is passed to a custom iBO page, CFID, CFToken, and XMLstring must be available to the custom iBO page, most likely as session variables.

CESerieBOAdmin.Restore(XMLString, CUser) returns a reference to CESeriesIntegration such that CESeriesIntegration will be set up as it was when GetXML was called. Any changes made to CUser, CContact, and CSubscription, and any products imported via the Basket tables or added via NewStdOrderLine, are present, as if control was passed from the instruction before GetXML to the instruction after Restore with no intervening activity.

To empty the basket tables from iBO, delete all of the CStdOrderLine instances associated with the CStdOrder and then call SaveToESeriesBasket.

If the user's ColdFusion session times out, the ColdFusion session variables are lost, including probably the XMLString. The user cannot continue their session because CFID and CFToken are no longer valid. Everything that the user has done that has not been committed to the database is lost. The next time the user logs in to the site, the basket table entries for the user will be deleted.

Basket tables

iBO assumes that the Basket tables will not be modified between the GetXML and Restore calls. Restore differs from LoadXML in the following ways:

- Restore is called on CESeriesBOAdmin
- Restore modifies the CUser instance, including DBID
- Restore returns a CESeriesIntegration instance

CESeriesBOAdmin.NewESeries uses the CFID and CFToken parameters that Restore obtains from the XML string. The XML string includes CESeriesIntegration data, CUser data, CContact, CAddress, CNote, CExtView, CSubscriptions, CSubscription, CStdOrder, and CStdOrderLine. The XML string does not include the Basket tables.

Viewing the iBO Change Log in Customer Portfolio

To view the iBO change log in Customer Portfolio, you must expand the data grid's row height or highlight the log text to see all of the logged fields created by iBO.

Usage Tips by Module

General Issues

Batch attributes

In *iMIS*, you can modify only the following batch attributes:

- batchnum
- data
- description
- control count

- control amount
- actual count
- cash table

iBO allows you to modify all batch attributes except for the `DateCreated` property.

International dates

If you are using international dates with iBO, you must configure the database and Web server to use the appropriate international locale. The locale on both the database and web server must be the same. Date information that is passed from the presentation layer to the iBO API must be in the same international locale format as the database and webserver. The iBO API returns date information to the presentation layer in this same international locale format.

iboCustomerManagement

Checksum for Automatically Assigned IDs

Checksum digits can be added to IDs that are automatically generated by *iMIS*. IDs generated with this option have a more unique value than IDs that are just sequentially assigned numbers. This only applies to IDs that are automatically generated for Name records; it does not apply to other IDs that are automatically generated.

Note: The checksum is not a random number. The checksum is derived from the ID using a specific set of algorithms.

To add checksums to automatically generated IDs, enable the **Use Checksum for IDs** option on the **Customer Setup - Basic Options** window (from **Customers**, select **Set up module> General**, and select **Basic Options**).

Synchronizing Phone Numbers and Email Addresses

iBO provides the same synchronization capabilities for Home Phone, Work Phone, Fax and E-mail as *iMIS* Customer Management.

The **Work Phone**, **Home Phone**, **Fax**, and **E-mail** fields (Name table) on the **Profile** tab on the **Manage customers** window can be mapped to the **Phone**, **Fax**, and **E-mail** fields (Name_Address table) on any address purpose defined in the **Main**, **2nd**, or **3rd** fields on the **Customer Setup - Address and Notes** window.

By mapping these fields, the data defined in these fields automatically populates the corresponding fields on the **Profile** tab or the address purpose tab (Name_Address data to Name data and vice versa). Synchronization allows data to match and eliminates the need to enter data twice.

See the *Customer Management* guide for more information on synchronizing phone numbers and e-mail addresses.

iboEvents

VAT and Canadian Taxation

iboEvents supports VAT and Canadian taxation.

- For VAT, if a meeting function is taxable and has a tax code assigned, all members will be charged VAT unless Name_Fin.USE_VAT_TAXATION is True.

- For Canadian taxation, if a tax authority is assigned at the meeting level and the meeting function is PST and/or GST taxable, all members will be charged tax unless Name_Fin.TAX_EXEMPT is not blank.

iboOrders

Registrations use iboOrders, which does not have any license key requirements. Other BOs and the ASI Presentation Layer code cannot call iboStdOrders if the "ORDER" key is not present.

If the ShipToID is changed while working with a CStdOrder, the new Ship-To contact is loaded from the database and verified. The address information for the CStdOrder is also refreshed to match the preferred mail address information of the new Ship-To contact.

Shipping Costs and Shipping Zones

Shipping costs will not be calculated correctly if an order is placed from a country for which shipping rates and zones have not been set up in *iMIS*.

Shipping zones are defined according to postal code. Depending on your organization's needs, you can define single postal codes as shipping zones or divide groups of postal codes into shipping zones.

Shipping zones use the first three characters of a postal code, and the table format for defining zones is *XXX,Z* where *XXX* represents the first three characters of a postal code, and Z represents the zone designation defined by the user. You also can enter a postal code in the format *XXX-XXX,Z* where *XXX* represents the first three characters of the beginning postal code, *XXX* represents the first three characters of the ending postal code, and *Z* represents the zone designation. These zone designations are used when setting up weight tables on the **Set up freight by weight** window.

For example, if you define a shipping zone for a single postal code, such as 75208, you would enter `752` followed by a comma. You then would enter either a numeric or alphabetic value for the zone designation, such as a `1`. The shipping zone would display as **752,1**. A shipping zone defined for a group of postal codes could be entered as `900-999,2`.

As long as you follow the correct table format, you can define as many zone codes as necessary. You must define shipping zone tables before defining freight tables. Zone tables are defined on the **Set up zones** window.

Quantity Shipped

The QtyShipped attribute in the iBO Order object is not set by iBO since nothing has yet shipped. The QtyShipped attribute is set by *iMIS* Orders when the order is shipped.

Multiple Entities for Products

iBO supports multiple entities at the product level. If multiple entities is enabled, the ORG_CODE is derived from the first non-blank product in the order. If there is no entity associated with the first non-blank product, the default order entity is used. If there is no order entity, the system default entity is used. If multiple entities is not enabled, the system default entity is used.

See the *Order Processing* guide for more information on multiple entities at the product level.

Code Samples

Sample scenarios and code

These sample code segments are for illustration only; be aware that they omit necessary conventions, such as releasing object instances and declaring and initializing variables.

Code sample: Contacts - Find and update data

The code below demonstrates the following scenario:

Find Contacts with ContactType = 'CM' and County = 'TRAVIS,' display them, and update the one with the ContactYears >= five, changing the Category to 'A'.

```
dim cboContactsAdmin as iboContactManagement.CContactsBOAdmin
dim contsContacts as iboContactManagement.Contacts
dim usrUser as iboUserSecurity.CUser
dim errErrors as iboErrors.Errors

usrUser.DBID = "ODBC:iMIS_dsn"

Set cboContactsAdmin = new CContacts.CContactsBOAdmin

Set contsContacts = cboContactsAdmin.GetContacts(usrUser)

contsContactsFilter = contsContacts.Filter

contsContactsFilter.Operator = efilAnd
contsContactsFilter.AddFilter(eContactType, "CM", efilEqual)
contsContactsFilter.AddFilter(eCounty, "TRAVIS", efilEqual)

contsContacts.GetContacts()

For x = 1 to contsContacts.Count
contContact = contsContacts.GetContact(x)
    strContactFirstName = contContact.FirstName
    etc.....
    (display Contact data row...)
    If contContact.ContactYears > 5 Then
        contContact.Category = 'A'
        contContact.Validate( )
        If contContact.ErrorCount > 0
            Set errErrors = contContact.Errors
            strError = errErrors.GetErrorMessage( )
            ' (process or display strError...)
        Else
            contContact.SubmitChanges(errErrors1,1)
        End If
    If contContact.ErrorCount > 0
        Set errErrors = contContact.Errors
        strError = errErrors.GetErrorMessage( )
        ' (process or display strError...)
    End If
 End If
Next x
```

Code sample: Contacts - Delete and insert data

The code below demonstrates the following scenario:

Retrieve Contact 'Scott Smith' who has ContactYears > 3, delete the address with purpose = "Temporary," add an address with purpose = "VacationHome," and display the addresses.

Note: For brevity, this code omits checking `Errors`.

```
contsContacts = cboContactsAdmin. NewContacts(usrUser)

contsContactsFilter = contsContacts.Filter
contsContactsFilter.Clear()

contsContactsFilter.Operator = efilAnd
contsContactsFilter.AddFilter(eLastName, "Smith", efilEqual)
contsContactsFilter.AddFilter(eFirstName, "Scott", efilEqual)
contsContactsFilter.AddFilter(eContactYears, "3", efilGT)

contsContacts.GetContacts()

If contsContacts.Count > 1 Then
    (whoops, got more than one member; handle it...)
Else
    contContact= CContacts.GetContact(1)
End If

contContact.GetAddresses()

For x = 1 to contContact.AddressCount
objAddress = contContact.GetAddress(x)
If objAddress.Purpose = "Temporary" Then
    objAddress.Delete()
End If
Next

addrAddress2 = contContact.NewAddress()
addrAddress2.Purpose = "VacationHome"
addrAddress2.Country = "Fiji"
addrAddress2....
CContact.SubmitChanges()
```

Code sample: Contacts - Update user-defined data

The code below demonstrates the following scenario:

Retrieve the user-defined data for a `Contact` and display the data from one of the user-defined windows, modify a field value, and submit the change.

```
contContact.GetExtViews()

evwExtView = contContact.GetExtView(1)

strViewName = evwExtView.ViewName
(display ViewName...)

evwExtView.ExtGetFields()   /* returns them in order by TabSeq (tab
order) */

For x = 1 to evwExtView.FieldsCount
efldField = evwExtView.ExtGetField(x)
strPrompt = efldField.Prompt
intFieldType = efldField.DataType
Select Case intFieldType
    Case iMISDataTypeChar
        intFieldWidth = efldField.FieldWidth
    Case iMISDataTypeInt
/* (Initialize as needed to prepare to display the field... ) */
    Case iMISDataTypeMoney
        ....
    Case iMISDataTypeCheckBox
        ....
    Case iMISDataTypeDate
        ....
    etc...
```

```
End Select
    (/* logic to display field prompt and value... */)
Next

/* (logic to retain/retrieve the field number corresponding to the
edited field...) */

efldField = evwExtView.GetField(x)
efldField.Value = /* (user input value) */
contContact.SubmitChanges()

/* process errErrors object... */
```

Code sample: Events - Display and register function data

The code below demonstrates the following scenario (assuming that the customer ID has been retrieved into the variable 'strCustID'):

Display available events with their function definitions, and register the customer's GUI selection of two functions.

```
dim cboEventsAdmin as iboEvents.CEventsBOAdmin
dim evtsEvents as iboEvents.CEvents
dim usrUser as iboUserSecurity.CUser

usrUser.DBID = "ODBC:imisdb"

Set cboEventsAdmin = new iboEvents.CEventsBOAdmin

' (Get events to choose from...)
Set evtsEvents = cboEventsAdmin.NewEvents(usrUser)
evtsEvents.GetEvents(True,True,True,100)
If evtsEvents.ErrorsCount > 0
' (handle errors by retrieving and parsing evtsEvents.Errors...)
End If

For x = 1 to evEvents.Count
    evtEvent = evEvents.GetEvent(x)
    strEventTitle = evtEvent.Title
    ' (display event title...)
    For y = 1 to evtEvent.FunctionsCount
        evtFunction = evtEvent.GetFunction(y)
        strDescription = evtFunction.Description
        ' etc... (display function information...)
    Next
Next

' (next page after event is chosen, and the EventCode is stored in
strEventCode, and the function codes are stored in strFunction1 and
strFunction2...)

Set regRegistration = cboEventsAdmin.NewRegistration(usrUser,
strEventCode, strRegistrantID,True,True,True,False)

regRegistration.NewLineItem(strFunctionCode1)
regRegistration.NewLineItem(strFunctionCode2)
If regRegistration.ErrorsCount > 0
' (handle errors by retrieving and parsing regRegistration.Errors...)
End If

regBadge = regRegistration.NewBadge()
regBadge.FirstName = "..."
regBadge.Designation = "..."
etc...
If regBadge.ErrorsCount > 0
    ' (handle errors by retrieving and parsing regBadge.Errors...)
End If

regPaymt = regRegistration.PaymentInfo
```

```
regPaymt.PayType = "CC"
regPaymt.CkOrCCNumber = "4111994822221111"
regPaymt.Amount = 495.00
regPaymt.CCExpire = "10/02"
regPaymt.CCType = "MC"
regPaymt.Validate
If regPaymt.ErrorCount > 0
' (handle errors by retrieving and parsing regPaymt.ErrorsObject...)
End If

regRegistration.SubmitChanges()
If regRegistration.ErrorCount > 0
' (handle errors by retrieving and parsing
regRegistration.ErrorsObject...)
End If
```

Code sample: Paging

This Visual Basic example will display all people with a last name in a given state, 10 at a time, sorted by last name. The first portion initiates paging and the second portion moves to the next page.

```
Option Explicit
Dim strPage As String
Dim Total As Integer
Dim curPage As Integer

Private Sub cmdFind_Select()
Dim I As Long
Dim CUser As iboUserSecurity.CUser
Dim cboa As iboContactManagement.CContactsBOAdmin
Dim contacts As iboContactManagement.CContacts
Dim contact As iboContactManagement.CContact
```

```
Set CUser = New iboUserSecurity.CUser
CUser.DBID = "ODBC:Student02"
Set cboa = New iboContactManagement.CContactsBOAdmin
Set contacts = cboa.NewContacts(CUser)
contacts.ContactFilter.Operator = efilAnd
contacts.ContactFilter.AddFilter \
ZiboContactManagement_CContact.StateProvince, _ \
    txtState, efilEqual
contacts.ContactFilter.AddFilter
ZiboContactManagement_CContact.LastName, _ \
    "", efilNotEqual
'NOTE THE NEW WAY OF SORTING - WORKS WITH PAGING - 'START WITH zibo
'USED IN FILTER TO SORT ON
'THEN ADD EITHER edrqAscending OR edrqDescending
'SO ODD NUMBER PARAMETERS ARE zibo, EVEN NUMBER 'PARAMETERS
edrqAscending OR edrqDescending
'YOU CAN HAVE AS MANY PAIRS AS YOU WANT, BUT MUST BE 'PAIRS
Call
contacts.ContactFilter.SetOrderList(ZiboContactManagement_CContact.LastN
ame, edrqAscending)
If contacts.ErrorsCount Then
    MsgBox contacts.Errors.ErrorsToString
End If
'NOTE - YOU MUST PERFORM A GET CONTACTS IN ORDER TO GET HOW MANY 'PAGES
ARE REQUIRED
'A PRODUCT ENHANCEMENT HAS BEEN MADE FOR THIS
Call contacts.GetContacts(True, 0)
Total = contacts.Count / 10
curPage = 1
strPage = contacts.GetContactsPaged("", 10, 1)
For I = 1 To contacts.Count
    Set contact = contacts.GetContact(I)
    List1.AddItem (contact.ContactID & " " & \
       contact.MainAddress.StateProvince _ \
       & " " & contact.LastName)
Next

End Sub
```

```
Private Sub cmdPage_Select()
    Dim CUser As iboUserSecurity.CUser
    Dim cboa As iboContactManagement.CContactsBOAdmin
    Dim contacts As iboContactManagement.CContacts
    Dim contact As iboContactManagement.CContact

    Set CUser = New iboUserSecurity.CUser
    CUser.DBID = "ODBC:Student02"
    Set cboa = New iboContactManagement.CContactsBOAdmin
    Set contacts = cboa.NewContacts(CUser)
    'YES - YOU MUST REDO THE FILTER EACH TIME YOU GET SUBSEQUENT
    'PAGES.  AFTER DISCUSSIONS WITH
    'DEVELOPERS, THEY COULDN'T CODE IT EASILY TO STORE THE FILTER,
    'SO YOU HAVE TO SPECIFY IT FOR
    'EACH SUBSEQUENT PAGE.  THIS MAY BE RESOLVED IN THE FUTURE.
    'THEY KNOW ABOUT THIS ISSUE.
    contacts.ContactFilter.Operator = efilAnd
    contacts.ContactFilter.AddFilter \
       ZiboContactManagement_CContact.StateProvince, _ \
       txtState, efilEqual
    contacts.ContactFilter.AddFilter \
       ZiboContactManagement_CContact.LastName, _ \
       "", efilNotEqual
    Call \
       contacts.ContactFilter.SetOrderList(ZiboContactManagement_\
       CContact.LastName, edrqAscending)

    If curPage < Total Then
        curPage = curPage + 1
    Else
        curPage = 1
        List1.Clear 'only clearing list when going back to 1st
        'page, so you can see it working
    End If
    strPage = contacts.GetContactsPaged(strPage, 10, curPage)
    Dim I As Integer
    For I = 1 To contacts.Count
        Set contact = contacts.GetContact(I)
        List1.AddItem (contact.ContactID & " " & \
           contact.MainAddress.StateProvince _ \
           & " " & contact.LastName)
    Next
End Sub
```

Sample application

The following Active Server Pages implementation (using Javascript) demonstrates how you could use the iBO interface to design a custom registration web application.

Note: To be implemented, this code requires some minor changes to the unencrypted *e*-Series source code. Contact ASI Consulting if you are interested in attempting a similar ASP implementation.

Code sample: index.asp

```
<HTML>
<HEAD>
<TITLE></TITLE>
<link rel='stylesheet'
href='http://eseriesdev/eseriesIBO/scriptcontent/stylesheet.css'
type='text/css'>
</HEAD>
<BODY>

<%
var strActivetab;
```

```
    /* Read in the querystring values... */
strActivetab = Request.QueryString("activetab");
strDSN = Request.QueryString("DSN");
strContactID = Request("ContactID");
strIP = Request("IP");
strEventCode = Request.QueryString("Event") ;
strFind = Request("Find");
strFilter = Request("Filter");

var strFilterTest = new String(Request("Filter"));

if (strFilterTest=="undefined"){
strFilter = 5;
}
%>

<table border="0" cellpadding="0" cellspacing="0" width="95%"><tr><td
align="left" style="padding: 0 0 0 0">

<!---tab table--->
<table border="0" cellpadding="3" cellspacing="0" class="tab-table">
    <tr>
<td align="middle" class=<% if (Request.QueryString("activetab")=="1"){
Response.Write("tab-active") }else{ Response.Write("tab-button")}%>><a
href="index.asp?DSN=<%=strDSN%>&IP=<%=strIP%>&ShowEvents=TRUE&ContactID=
<%=Request("ContactID")%>&activetab=1">Select Event</a></td>
<td align="middle" class=<% if (Request.QueryString("activetab")=="2"){
Response.Write("tab-active") }else{ Response.Write("tab-button")}%>><%
if (Request.QueryString("activetab")=="2" ||
Request.QueryString("activetab")=="3"){Response.Write("<a
href=index.asp?DSN="+strDSN+"&ShowFunctions=TRUE&IP=" + strIP +
"&ContactID="+Request("ContactID")+"&activetab=2&Event="+strEventCode+">
Event</a>");} else {Response.Write("<span
class=EN10>Event</span>");}%></td>
<td align="middle" class=<% if (Request.QueryString("activetab")=="3"){
Response.Write("tab-active") }else{ Response.Write("tab-
button")}%>><span class="EN10">Event Detail</SPAN></td>
<td align="middle" class=<% if (Request.QueryString("activetab")=="4"){
Response.Write("tab-active") }else{ Response.Write("tab-
button")}%>><span class="EN10">Event Review</SPAN></td>
    </tr></table>
<!---/tab table--->

</td></tr>
<tr><td align="left" style="padding: 0 0 0 0">

<!---tab body table--->
<table border="0" cellpadding="3" cellspacing="0" class="tab-table"
width="100%">
<form
action="index.asp?DSN=<%=strDSN%>&ShowEvents=TRUE&ContactID=<%=Request("
ContactID")%>&activetab=1" method=post>
<tr class="tab-button"><td colspan="4" align="right"><a
href=http://srobinson-w2k/eseriesIBO>Back to Main Site</a> | <span
class=EN10>Find events that</span> <select name="Filter"><option
value=5>Contain</option><option value=6>Begins With</option></select>
<input type="text" name="find" size="10" value="<%=(strFind)%>"> <input
type="submit" value="go"></td></tr>
    </form>
    <tr bgcolor="black" height="1"><td colspan="4"></td></tr>
    <tr><td colspan="4">
    <!------>

<TABLE cellSpacing=0 cellPadding=3 border=0 width=95% align="center">
 <TR>
<%
       var EvtAdminObject ;
       var EvtsObject ;
```

```
       var EvtObject ;
       var UserObject ;
       var lngCount ;
       var errObject ;
       var strLink1 ;
       var strLink ;

/* The following will be enumerated constants in the future... */
       var lngEvtPropMeeting = 2001001 ;
       var lngEvtPropTitle = 2001004 ;

       var intfilEqual = 1 ;
       var intfilNotEqual = 2 ;
       var intfilGT = 3 ;
       var intfilLT = 4 ;
       var intfilContains = 5 ;
       var intfilBeginsWith = 6 ;

/* Set up a User and Events admin object to instantiate IBO Events. */
       UserObject = Server.CreateObject("iboUserSecurity.CUser") ;
       UserObject.DBID = "ODBC:" + Request.QueryString("DSN") ;
EvtAdminObject = Server.CreateObject("iboEvents.CEventsBOAdmin") ;
       EvtsObject = EvtAdminObject.NewEvents(UserObject) ;

if(Request.QueryString("ShowEvents")=="TRUE")
/* Displaying the events list... */
       {
/* Filter, retrieve, and display events based on the supplied filter...
*/
Response.Write("<TR><TD COLSPAN=2><span class=EB10>Select an event to
begin registration</span></td></tr>") ;

strLink1 = "<a href=http://srobinson-
w2k/eseriesIBO/ibo_demo/index.asp?activetab=2&DSN=" + strDSN + "&IP=" +
strIP + "&ContactID=" + strContactID + "&ShowFunctions=TRUE&Event=" ;

EvtsObject.EventFilter.AddFilter(eEvtTitle, strFind, strFilter)  ;
/* params: true - clear collection first, false - don't retrieve
function data, false - don't retrieve function fee data, 100 maximum
records */
       EvtsObject.GetEvents(true,false,false,100) ;

           lngCount = 1 ;
Response.Write("<tr class=ETH><td></td><td colspan=2
height=1></td></tr>") ;
           while(lngCount <= EvtsObject.Count)
    /* The events are displayed such that they will link to the
ShowFunctions page (below) with the eventcode in the QueryString. */
EvtObject = EvtsObject.GetEvent(lngCount) ;
strLink = "<tr><td align=right valign=middle class=ETD width=150>" +
strLink1 + EvtObject.EventCode + ">" + EvtObject.Title + "</a></td><td
align=left valign=middle class=ETDALT width=425><span class=EN10>" +
EvtObject.Description + "</span></td></tr>"
Response.Write(strLink) ;
Response.Write("<tr class=ETH><td></td><td colspan=2
height=1></td></tr>") ;
               lngCount += 1 ;
           }

Response.Write("") ;
}

if(Request.QueryString("ShowFunctions") == "TRUE")
/* Displaying the functions for a selected event... */
{

var strChkBox ;
var strDSN ;
var strEventCode ;
```

```
strEventCode = Request.QueryString("Event") ;

strLink1 = "   <a href=http://srobinson-
w2k/eseriesIBO/ibo_demo/index.asp?DSN=" + strDSN + "&IP=" + strIP +
"&Event=" + strEventCode + "&ContactID=" + strContactID +
"&FunctionDetail=TRUE&activetab=3&Function=" ;

EvtObject = EvtsObject.GetEventByCode(strEventCode) ;

/* Now display the functions on a submit form, each with a checkbox...
*/
Response.Write("<FORM ACTION=http://srobinson-
w2k/eseriesIBO/ibo_demo/RegistrationReview.asp?activetab=4&Event=" +
strEventCode + "&ContactID = " + strContactID + " METHOD=POST/>") ;

Response.Write("<TD>") ;
Response.Write("<span class=EB10>Event: " + EvtObject.Title +
"</SPAN><BR>") ;
Response.Write("<span class=EN8>Select register select one or more
functions</SPAN>") ;
Response.Write("</TD></TR>") ;

       lngCount = 1 ;

Response.Write("<INPUT TYPE=HIDDEN NAME=Event VALUE=" + strEventCode +
">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=ContactID VALUE=" + strContactID
+ ">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=DSN VALUE=" + strDSN + ">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=ACTIVETAB VALUE=4>") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=IP VALUE=" + strIP + ">") ;
while(lngCount <= EvtObject.FunctionsCount)
    /* Each function will be displayed with the event and function code
in the querystring, linking to the Function Detail page (below)... */
           Response.Write("<TR><TD>") ;
           FunctionObject = EvtObject.GetFunction(lngCount) ;
strLink = strLink1 + FunctionObject.FunctionCode + ">" +
FunctionObject.FunctionTitle + "</a>"
strChkBox = " <INPUT TYPE=CHECKBOX VALUE=ON NAME=" +
FunctionObject.FunctionCode + " style=border-width=0;>"
Response.Write(strChkBox + strLink) ;

           lngCount += 1 ;
           Response.Write("</TD></TR>") ;
}

       Response.Write("<TR><TD>") ;

Response.Write("<SPAN CLASS=EN10>Enter Registrant Class Code: <INPUT
TYPE=TEXT NAME=RegClass SIZE=8></SPAN> ") ;
       Response.Write(" <INPUT TYPE=SUBMIT VALUE=Review> ") ;

       Response.Write("</TD>") ;
       }

if(Request.QueryString("FunctionDetail") == "TRUE")
/* Displaying the detail for a selected function... */
    {
       var EvtAdminObject ;
       var EvtsObject ;
       var EvtObject ;
       var UserObject ;
       var FunctionObject ;
       var errObject ;
       var strFunctionCode ;
       var strAutoEnroll ;

strEventCode = Request.QueryString("Event") ;
```

```
strFunctionCode = Request.QueryString("Function") ;

UserObject = Server.CreateObject("iboUserSecurity.CUser") ;
UserObject.DBID = "ODBC:" + Request.QueryString("DSN") ;
EvtAdminObject = Server.CreateObject("iboEvents.CEventsBOAdmin") ;
EvtsObject = EvtAdminObject.GetEventsObj(UserObject) ;

EvtObject = EvtsObject.GetEventByCode(strEventCode) ;
FunctionObject = EvtObject.GetFunctionByCode(strFunctionCode) ;

if(FunctionObject.AutoEnroll)
           strAutoEnroll = "YES" ;
       else
           strAutoEnroll = "NO" ;

       Response.Write("<TD>") ;
Response.Write("<span class=EB11>Event: " + EvtObject.Title +
"</SPAN><BR>") ;
Response.Write("<span class=EB11>Function: " +
FunctionObject.FunctionTitle + "</SPAN>") ;
Response.Write("<br><P align=center><span class=EN10><U>Function
Detail:</U></SPAN><br><BR>") ;
       strPct = "%"
Response.Write("<TABLE align=center width=100"+strPct+">") ;

/* Function Detail : */
strHeaderBegin = "<TD align=right><span class=EB10>" ;
strHeaderEnd = ": </SPAN></TD>"
strValueBegin = "<TD align=left><span class=EN10>"
strValueEnd = "</SPAN></TD>"

Response.Write("<TR>") ;
Response.Write(strHeaderBegin + "Function Code" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.FunctionCode +
strValueEnd) ;
Response.Write(strHeaderBegin + "Title" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.FunctionTitle +
strValueEnd) ;
Response.Write("</TR>") ;
Response.Write("<TR>") ;
Response.Write(strHeaderBegin + "Begin Date" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.BeginDate + strValueEnd) ;
Response.Write(strHeaderBegin + "End Date" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.EndDate + strValueEnd) ;
Response.Write("</TR>") ;
Response.Write("<TR>") ;
Response.Write(strHeaderBegin + "Maximum Attendence" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.MaxAttendees +
strValueEnd) ;
Response.Write(strHeaderBegin + "Auto-Enroll?" + strHeaderEnd) ;
Response.Write(strValueBegin + strAutoEnroll + strValueEnd) ;
Response.Write("</TR>") ;
       Response.Write("<TR>") ;
Response.Write(strHeaderBegin + "Status" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.Status + strValueEnd) ;
Response.Write(strHeaderBegin + "Total Registered" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.RegistrantsCount +
strValueEnd) ;
Response.Write("</TR>") ;
Response.Write("<TR>") ;
Response.Write(strHeaderBegin + "Type" + strHeaderEnd) ;
Response.Write(strValueBegin + FunctionObject.FunctionType +
strValueEnd) ;
Response.Write("</TR>") ;

       Response.Write("</TABLE>") ;

Response.Write("<br><SPAN CLASS=EN10><U>Registrant Fees for this
function:</U></SPAN><br><br>") ;
```

```
Response.Write("<TABLE cellpadding=3 cellspacing=0 align=center
width=100"+strPct+">") ;

       lngCount = 1 ;

Response.Write("<TR><TD CLASS=ERSHeader align=center
valign=middle>Registrant Class</TD>") ;
Response.Write("<TD CLASS=ERSHeader align=center
valign=middle>Code</TD>") ;
Response.Write("<TD CLASS=ERSHeader align=center
valign=middle>Complementary?</TD>") ;
Response.Write("<TD CLASS=ERSHeader align=center valign=middle>Early
Fee</TD>") ;
Response.Write("<TD CLASS=ERSHeader align=center valign=middle>Regular
Fee</TD>") ;
Response.Write("<TD CLASS=ERSHeader align=center valign=middle>Late
Fee</TD></TR>") ;
while(lngCount <= FunctionObject.FunctionFeesCount)
    /* Function fee detail for each fee definition... */
ffeeFunctionFee = FunctionObject.GetFunctionFee(lngCount) ;
       strRegClass = ffeeFunctionFee.RegClassDescription ;
       strRegClassCode = ffeeFunctionFee.RegClass ;
       strFee1 = "$" + ffeeFunctionFee.EarlyFee ;
       strFee2 = "$" + ffeeFunctionFee.RegularFee ;
       strFee3 = "$" + ffeeFunctionFee.LateFee ;
       strIncomeAcct = ffeeFunctionFee.IncomeAccount ;
       if (ffeeFunctionFee.Complementary == true)
              strComplementary = "YES" ;
           else
              strComplementary = "NO" ;

Response.Write("<TR><TD><I><span class=EN10>" + strRegClass +
"</I></span></TD>") ;
Response.Write("<TD><span class=EN10>" + strRegClassCode +
"</span></TD>") ;
Response.Write("<TD align=center valign=middle><span class=EN10>" +
strComplementary + "</span></TD>") ;
Response.Write("<TD><span class=EN10>" + strFee1 + "</span></TD>") ;
Response.Write("<TD><span class=EN10>" + strFee2 + "</span></TD>") ;
Response.Write("<TD><span class=EN10>" + strFee3 + "</span></TD></TR>")
;

           lngCount += 1 ;
       }

Response.Write("</TABLE>") ;
}
%>
</TR></TABLE>
<!------>

</td></tr></table>
<!---/tab body table--->

</td></tr></table>

<img scr="../images/1x1.gif" width="1" height="10"><br>
<table border="0" cellpadding="3" cellspacing="0" class="tab-table"
width="95%">
<tr bgcolor="black" height="1"><td colspan="3"></td></tr>
<tr class="tab-button"><td colspan="3" align="right">webmaster</td></tr>
<tr><td colspan="3"> </td></tr></table>

</HTML>
```

Code sample: RegistrationReview.asp

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
<link rel='stylesheet'
href='http://eseriesdev/eseriesIBO/scriptcontent/stylesheet.css'
type='text/css'>
</HEAD>
<BODY>

<%
var strActivetab;
strActivetab = Request.QueryString("activetab");
strDSN = Request.QueryString("DSN");
strContactID = Request("ContactID");
strEventCode = Request.QueryString("Event") ;
strIP = Request("IP") ;

%>

<table border="0" cellpadding="0" cellspacing="0" width="95%"><tr><td
align="left" style="padding: 0 0 0 0">

<!---tab table--->
<table border="0" cellpadding="3" cellspacing="0" class="tab-table">
    <tr>
<td align="middle" class=<% if (Request.QueryString("activetab")=="1"){
Response.Write("tab-active") }else{ Response.Write("tab-button")}%>><a
href="index.asp?DSN=<%=strDSN%>&ShowEvents=TRUE&IP=<%=strIP%>&ContactID=
<%=Request("ContactID")%>&activetab=1">Select Event</a></td>
<td align="middle" class=<% if (Request.QueryString("activetab")=="2"){
Response.Write("tab-active") }else{ Response.Write("tab-button")}%>><%
if (Request.QueryString("activetab")=="2" ||
Request.QueryString("activetab")=="3" ||
Request.QueryString("activetab")=="4"){Response.Write("<a
href=index.asp?DSN="+strDSN+"&ShowFunctions=TRUE&IP=" + strIP +
"&ContactID="+Request("ContactID")+"&activetab=2&Event="+strEventCode+">
Event</a>");} else {Response.Write("<span
class=EN10>Event</span>");}%></td>
<td align="middle" class=<% if (Request.QueryString("activetab")=="3"){
Response.Write("tab-active") }else{ Response.Write("tab-
button")}%>><span class="EN10">Event Detail</SPAN></td>
<td align="middle" class=<% if (Request.QueryString("activetab")=="4"){
Response.Write("tab-active") }else{ Response.Write("tab-
button")}%>><span class="EN10">Event Review</SPAN></td>
    </tr></table>
<!---/tab table--->

</td></tr>
<tr><td align="left" style="padding: 0 0 0 0">

<!---tab body table--->
<table border="0" cellpadding="3" cellspacing="0" class="tab-table"
width="100%">
<form
action="index.asp?DSN=<%=strDSN%>&ShowEvents=TRUE&ContactID=<%=Request("
ContactID")%>&activetab=1" method=post id=form1 name=form1>
<tr class="tab-button"><td colspan="4" align="right"><a
href=http://srobinson-w2k/eseriesIBO>Back to Main Site</a> | <span
class=EN10>Find events that</span> <select name="Filter"><option
value=5>Contain</option><option value=6>Begins With</option></select>
<input type="text" name="find" size="10"> <input type="submit"
value="go" id=submit1 name=submit1></td></tr>
    <tr bgcolor="black" height="1"><td colspan="4"></td></tr>
    </form>
    <tr><td colspan="4" style="padding: 0 0 0 20">
```

```
 <TABLE cellSpacing=0 cellPadding=3 border=0 width="45%">
      <TR>

       <!-- Body: -->
       <%

       var EvtAdminObject ;
       var EvtsObject ;
       var EvtObject ;
       var UserObject ;
       var lngCount ;
       var blnError ;
       var strLink1 ;
       var strLink ;
       var strChkBox ;
       var strDSN ;
       var strEventCode ;
       var strFunctionCode ;
       var strRegClass ;
       var funcFee ;
       var Registration ;
       var strContactID ;
       var blnInit ;
       var strConfirmed ;
       var lngItems ;

/* This page is either being posted to from the Functions selection
form, or it is being posted to from itself, after the user has reviewed
the registration and decides to submit it. First, it sets up a
registration object and populates it according to whatever was selected
on the Functions selection form (calling Registration.NewLineItem() for
each function). Then it either submits the registration if this page was
posted to after review, or it displays the registration information for
review (the form displays and the submit button posts back to this
page). */

       blnError = false ;

    UserObject = Server.CreateObject("iboUserSecurity.CUser") ;
       UserObject.DBID = "ODBC:" + Request("DSN") ;
EvtAdminObject = Server.CreateObject("iboEvents.CEventsBOAdmin") ;
    EvtsObject = EvtAdminObject.NewEvents(UserObject) ;
       strPct = "%"
Response.Write("<FORM ACTION=http://srobinson-
w2k/eseriesIBO/ibo_demo/RegistrationReview.asp? METHOD=POST id=form1
name=form1>") ;
    Response.Write("<TD" + ">") ;

    strEventCode = Request("Event") ;
    strRegClass = Request("RegClass") ;
    strContactID = Request("ContactID") ;
    strConfirmed = Request("Confirmed") ;

    EvtObject = EvtsObject.GetEventByCode(strEventCode) ;
Response.Write("<img scr=../images/1x1.gif width=1 height=3><br>");

Response.Write("<span class=EB10>Event: " + EvtObject.Title + "</span>")
;

Response.Write("<INPUT TYPE=HIDDEN NAME=Event VALUE=" + strEventCode +
">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=ContactID VALUE=" + strContactID
+ ">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=Confirmed VALUE=TRUE>") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=DSN VALUE=" + strDSN + ">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=RegClass VALUE=" + strRegClass +
">") ;
Response.Write("<INPUT TYPE=HIDDEN NAME=IP VALUE=" + strIP + ">") ;
```

```
    lngCount = 1 ;
        lngItems = 0 ;
while(lngCount <= EvtObject.FunctionsCount && blnError == false)
        {
            FunctionObject = EvtObject.GetFunction(lngCount) ;

            blnInit = false ;
            strFunctionCode = FunctionObject.FunctionCode ;

            if (Request(strFunctionCode) == "ON")
            {
                if(lngItems == 0)
                {
/* Parameters for the following: UserObject contains database context,
EventCode and ContactID for the registration, false - don't auto-enroll
any automatic functions, false - don't attempt to auto-enroll any linked
functions, true - initialize the registration with default pricing, etc.
and true - this is an e-Series basket item. */
Registration =
EvtAdminObject.NewRegistration(UserObject,strEventCode,strContactID,fals
e,false,true,true) ;
    if(EvtAdminObject.ErrorsCount > 0)
Response.Write("<br><br><span class=EB10>ERROR: " +
EvtAdminObject.Errors.GetErrorMessage() + "</span>") ;
                                blnError = true ;
                        }
                        else
                                Registration.RegClass = strRegClass ;

                }
                if(blnError==false)
                {       lngItems += 1 ;
Registration.NewLineItem(strFunctionCode,false) ;
/* false param: don't add linked functions */
Response.Write("<INPUT TYPE=HIDDEN NAME=" + strFunctionCode + "
VALUE=ON>") ;
                }
            }

            lngCount += 1 ;
        }

    if(strConfirmed == "TRUE" && blnError==false)
        {
            Registration.BatchNum = "93" ;
            Registration.IPAddress = strIP ;
            Registration.SubmitChanges() ;
            if(Registration.ErrorsCount==0)
            {
Response.Write("<P align=center>Registration was successfully
submitted.</p>") ;
Response.Write("<P align=center><a
href=http://eseriesdev/eseriesibo/source/Meetings/cBasketCheckOut.cfm>Pr
oceed to Basket</a></p>")
            }
            else
Response.Write(Registration.Errors.GetErrorMessage()) ;
        }
        else if(blnError==false)
        {
Response.Write("<br><span class=EN8>Please Review Your
Registration</span>") ;
Response.Write("<br><img scr=../images/1x1.gif width=1 height=10
border=0><br>");
if(lngItems > 0)
{
Response.Write("<TABLE align=center cellpadding=3 cellspacing=0 border=0
width=95"+strPct+">") ;
```

```
Response.Write("<TR><TD class=ERSHeader align=center
valign=middle>Function</TD>") ;
Response.Write("<TD class=ERSHeader align=center
valign=middle>Price</TD></TR>") ;

              lngCount = 1 ;
       while(lngCount <= Registration.LineItemsCount)
              {
Response.Write("<TR><TD><span class=EN9>" +
Registration.GetLineItem(lngCount).FunctionTitle + "</span></TD>") ;
Response.Write("<TD><span class=EN9>$" +
Registration.GetLineItem(lngCount).ExtendedAmount + "</span></TD></TR>")
;
Response.Write("<TR><TD COLSPAN=2 HEIGHT=1 BGCOLOR=BLACK></TD></TR>");
           lngCount += 1 ;
              }
Response.Write("<TR><TD align=right><span
class=EN9>Total:</span></TD><TD><span class=EN9>$" +
Registration.TotalCharges + "</span></TD></TR>") ;
Response.Write("</TABLE><TABLE align=center><TR><TD><INPUT TYPE=SUBMIT
VALUE=SAVE></TD></TR></TABLE>") ;
           }
           else
       Response.Write("<P align=center>(No Functions Selected)") ;
       }
    Response.Write("</TD>") ;

    %>

    </TR>
</TABLE>
<img scr=../images/1x1.gif width=1 height=3><br>

</td></tr></table>
<!---/tab body table--->

</td></tr></table>
<table border="0" cellpadding="3" cellspacing="0" class="tab-table"
width="95%">
<tr bgcolor="black" height="1"><td colspan="3"></td></tr>
<tr class="tab-button"><td colspan="3" align="right">webmaster</td></tr>
<tr><td colspan="3"> </td></tr></table>

</BODY>
</HTML>
```

Glossary of Terms

API

Application Program Interface. A language and message format used by an application to communicate with, in this case, the database management system behind *iMIS*. You implement APIs by writing function calls in the program, which provide the linkage to the required subroutine for execution.

ASP

Active Server Pages, Microsoft's webserver technology. Not to be confused with the other common meaning: Application Service Provider.

Business Object

A class providing a way to access to logical data within the *iMIS* database. A Component supports one or more such classes.

Class

Programming unit of software within a component. Each class implements at most one interface so that scripting languages can be supported.

ColdFusion

This is the webserver technology that *e*-Series uses to create dynamic web pages. Allaire is the producer of this award winning software, which was acquired by the software leader Macromedia in 2001.

COM Components

Microsoft's branding for files with a .DLL extension, as well as the Microsoft technologies that support scalable server use of these files.

Component

Executable unit of software, deployed as a dynamically linked library (DLL). A component supports one or more business object interfaces. One or more classes implement each component.

Configure

Setting an *e*-Series option. For instance the Systems_Params group of options. Also adjusting the cascading style sheet or *e*-Series module_config .txt files. Configuration can also include editing HTML in the `/ScriptContent` group of files.

Customize

This refers to making a change in *e*-Series that cannot currently be done by configuring one of *e*-Series options. The only customizations that ASI encourages are customizations developed using iBO. If iBO is used, then when upgrading, the analysis, recoding, retesting and re-approval process is minimal, allowing clients to upgrade with the same ease as with ASI's *iMIS* product.

DBID

Database Identifier, a property of the User object that provides the database context for methods that you are calling.

Dynamic Web Page

A web page whose contents vary depending upon parameters provided to the webserver. The content that varies is read from a database based upon these parameters. *e*-Series generates dynamic web pages.

Encrypted Files

The bulk of the *e*-Series scripts, which reside under the `/Source` folder. These files are encrypted so that consultants cannot adjust these scripts. Adjusting these scripts affects a clients upgrade path, as all customizations performed to these scripts must be analyzed, and possibly recoded, tested and re-approved by the client as part of the upgrade to a future release of *e*-Series.

iBO

iMIS Business Objects, the library of business logic that enables you to extend *iMIS* with an interface and workflow of your own design.

iMIS

The *iMIS* application suite, including its SQL database.

Interface

Reference to an instance of a class within a component. A class is used to implement an interface. Multiple classes can implement the same interface if they are in different components. This is an example of polymorphism using interfaces.

Module_Config

These unencrypted files hold the label settings used in *e*-Series.

ScriptContent

Refers to any of the ColdFusion files that reside in the `/ScriptContent` folder. None of these files is encrypted, so consultants may configure any of these files.

Index

The page numbers in these index entries indicate the page on which the relevant topic heading begins, not the page on which the keyword appears.

D

E

F

G

T

U

V

W

X