



# Retail Pro 9

## Plugin Programmer's Guide

Retail Pro International, LLC  
400 Plaza Dr., Suite 200  
Folsom, CA 95630 USA

USA 1-800-738-2457  
International +1-916-605-7200  
[www.retailpro.com](http://www.retailpro.com)



## About this Guide

This document explains how to create Plugins (customizations) for Retail Pro 9.

While Retail Pro International, LLC. endeavors in good faith to ensure that our documentation is an accurate reflection of the capabilities of our products, it is possible that certain inadvertent and/or typographical errors may nonetheless occur, in spite of our best efforts to avoid the same. If you as a licensed user of these product(s) encounter what appears to be an inconsistency in the documentation as compared to your perception of the performance of the product(s), please contact your Retail Pro Authorized Business Partner and inform them promptly of the discrepancy so they/we can have the opportunity to correct the documentation. In addition, while our documentation is generally a current reflection of the actual features and operation of our products, changes to the features and operation of our products which occur between formal releases of the documentation can be found on the [my.retailpro.com](http://my.retailpro.com) website, in the Documentation Portal, in the Release Notes area, and any such changes are incorporated herein by reference.

**Retail Pro International, LLC  
400 Plaza Dr., Suite 200  
Folsom, CA 95630 USA**

**USA 1-800-738-2457  
International +1-916-605-7200  
[www.retailpro.com](http://www.retailpro.com)**

---

Copyright © 2019 All rights reserved. Retail Pro International, LLC.

USA 1-800-738-2457  
International +1-916-605-7200  
[www.retailpro.com](http://www.retailpro.com).

### Trademarks

Retail Pro and the Retail Pro logo are registered trademarks and/or registered service marks of Retail Pro, Inc. in the United States and other countries. Oracle and Oracle 11g are registered trademarks and/or registered service marks of Oracle Corporation. All rights reserved. Other parties' trademarks or service marks are the property of their respective owners and should be treated as such.

### Document Revision History

<i>Date</i>	<i>Notes</i>
04/25/2017	Added "API Updates" section.
01/10/2018	Update of IabstractPlugin.GUID
02/01/2019	Merged EFT Plugin API legacy doc with Plugin Programmer's Guide (#34285)

## Table of Contents

About this Guide .....	iii
API Updates (Chronological Order) .....	6
Introduction .....	7
About Using Plugins .....	7
Plugin Components and Life Cycle .....	10
Key Components of a Plugin .....	10
Plugin Life Cycle .....	10
Business Objects .....	14
Business Object Wrappers .....	17
TenderBO Wrapper .....	18
Plugin Manager .....	28
Plugin Adapters .....	29
Notifications/Events .....	30
Creating a Plugin in Delphi .....	36
Setting Up Your Development Environment .....	36
Creating a Plugin - Basic Steps .....	36
Importing the Plugins Type Library (Plugins.tlb) .....	37
Creating a New COM Server Object .....	37
Creating a New Type Library .....	38
Creating the Plugin Unit .....	45
Creating the Plugin Classes .....	45
Creating the Discovery Class .....	53
Exporting Procedures .....	57
Registering the COM Servers .....	57
Manifest Files .....	58
Test Your Plugin .....	59
Error Handling .....	62
Tips and Tricks .....	63
Hardware Plugins .....	67
Tender and EFT Plugins .....	68
Retail Pro 9 Hardware Interface Implementations .....	70
IDisplayPlugin .....	70
ICashDrawerPlugin .....	70
IMSRPlugin .....	71
ICheckImageScannerPlugin .....	71
IMICRPlugin .....	72
IPinPadPlugin .....	73
IScalePlugin .....	73
IBarCodeScannerPlugin .....	74
IInventoryScannerPlugin .....	74
IFiscalPrinterPlugin .....	74
User-Interface Data Access .....	89
Plugins Type Library (Plugins.tlb) .....	91

Interfaces .....	91
The Plugin Adapter Interface .....	94
IPluginAdapter .....	94
Plugin Licensing .....	149
ILicense .....	149
Parent Interfaces .....	151
IAbstractPlugin .....	151
IBOPlugin .....	152
IAttributePlugin .....	153
IHardwarePlugin .....	154
Base Interfaces .....	155
IAttributeValueValidationPlugin .....	155
IAttributeAssignmentPlugin .....	155
IEntityUpdatePlugin .....	155
IItemAddRemovePlugin .....	158
ISideButtonPlugin .....	161
ICustomAttributePlugin .....	162
IPrintPlugin .....	162
ITenderPlugin .....	162
IEFTPlugin .....	164
ITenderDialoguePlugin .....	165
CustomPluginClass .....	173
Enumerated Values .....	179
Preference Constants .....	183
Distributing Plugins .....	201
Distributing Plugins .....	201
Appendix A. Retail Pro Fields and Constants .....	202
Tender Types .....	202
EFT Result Codes .....	202
CVV2 Result Codes .....	202
AVS Result Codes .....	202
Boolean Constants .....	203
EFT Action Codes (Awaiting Revision) .....	203
Name=Value Name Constants .....	204
Appendix B. Pseudo/Pascal Code Samples .....	207
Simple Sale Transaction .....	207
Sale with a Void Transaction .....	207
Index .....	209

## API Updates (Chronological Order)

<i>Release</i>	<i>Date</i>	<i>Change</i>
9.40.4.099 9.30.9.477 9.40.3.98 (CCU)	2/06/2017	Modified ITenderDialoguePlugin and ITenderDialoguePluginDisp to have additional accessible fields for new EFT database fields.
9.40.4.100 9.30.9.477 9.40.3.98 (CCU)	2/01/17	Modified uTenderBOWrapper to include expanded new EFT fields.
9.40.3.077 9.30.8.461	11/16/2016	Added new ADDLINEITEMto InvoiceBOWrapper ExecuteMethod to allow a plugin to add a line item to a document via the item sid
9.40.2.041 9.30.7.438	8/18/2016	Added BusinessObjectType btSlipItemBO
9.40.2.050 9.30.7.452	9/15/2015	Added new SETGCVALUES to InvoiceBOWrapper ExecuteMethod to allow a plugin to set the value of a gift card without impacting its price
9.30.2.215 9.20.770.557	6/24/2015	added permission bpReqCommInZOutClose
9.30.2.167	3/18/2015	Added new SETGCVALUES to InvoiceBOWrapper ExecuteMethod to allow a plugin to set the value of a gift card without impacting its price
9.30.2.147 9.30.1.153 (CCU)	2/09/2015	Added new RECALCTAX option to InvoiceBOWrapper ExecuteMethod to recalculate the taxes on items. This is in addition to the existing options of CENTRALPROCESSREQUEST, PERFORMCASHDROP, SPLITLINEITEM. and PRINTINFORECEIPT
9.30.1.130 9.20.769.798	1/09/2015	EFTCapabilities had new options added ecCCAAuthSettle ecDCAuthSettle ecGCAuthSettle ecCheckAuthSettle
9.30.0.113 9.20.769.490	10/15/2014	EFTCapabilities hadnew option added ecCanUseGatewayOffline
9.30.0.90 9.20.768.454	5/29/2014	Added InvnScannerCapability with the one option of iscapUseWSSubdir

Release	Date	Change
9.20.764.354	5/02/2013	added permissions bpCentGCAAllowCardToBeKeyed ; bpCentGCMaskInUI bpTrimAluInPILookup bpNextInvSeqNo_NoUpdate bpNextReturnInvSeqNo_NoUpdate bpNextCashDropSeqNo_NoUpdate bpNextDepositSeqNo_NoUpdate
9.30.085 9.20.764.344	3/29/2013	EFTCapabilities had two new options added ecUseMerchantWarehouse ecUseGeniusDevice
9.30.0.087 9.20.764.346 9.20.763.352 (CCU)	3/29/2013	Added LineDisplaycapabilities with the one option of ldcapMerchantWareFormat

## Introduction

### About Using Plugins

Retail Pro business partners often require additional functionality that is specific to their customer base, or that must be supplied in a shorter time frame than Retail Pro can accommodate. This functionality is supplied by *Plugins*.

You can easily create a Plugin that extends the functionality of Retail Pro while enforcing the business and security rules required by the core product. Retail Pro does not limit your choice of development platform and architecture and provides the greatest flexibility and support for current technologies. You can create Plugins using any of the standard development platforms, such as Visual Basic, C, C++, or C#.

#### Restrictions

- The API architecture for Retail Pro 9.x is different than in previous versions or Retail Pro. Plug-ins are no longer Borland Delphi BPL's but utilize a COM architecture.
- Plug-ins will be DLL's that will implement one or more interfaces to act as a COM service for Retail Pro.
- Retail Pro will have no direct interaction with gateways or processors. All communication to and from Retail Pro will be through the plug-in.
- Plug-ins will be responsible for providing all data transformation and any gateway specific UI.
- Plug-ins will be responsible for providing event-based logging mechanisms for debugging purposes.
- Plug-ins will be responsible for maintaining independent configuration data and providing UI for manipulating the plug-in configuration.

## Types of Plugins

There are two basic types of Plugins: **Hardware** and **User Interface**.

- Hardware Plugins are used for specific POS hardware, such as line displays, cash drawers, MICR readers, etc. The Retail Pro Plugin API mimics most of the properties and method signatures for the device interfaces described by the UPOS specification. (See <http://www.monroeecs.com/unifiedpos.htm> for details on the UPOS specification.)
- User interface Plugins supplement the functions of the user interface, such as adding a button to a menu or validating a field value.

## What is a Plugin and How Does It Work?

A COM server is a DLL constructed to be registered and work with Window's COM services. A Plugin is a special class contained in a COM server module. There can be one or more Plugin classes in a COM server module, but there must always be a "discovery" class in each server module. All Plugin classes must implement one of the Plugin interfaces described in the Plugins type library (Plugins.tlb).

The COM server DLL is registered with Windows using regsvr32. The Plugin classes implement one or more of the interfaces in the Plugins type library. The COM server will also define its own type library in which a "coclass" is defined for each Plugin class. The GUID for a coclass is stored in the registry when the DLL is registered with Windows, and its GUID is used to uniquely identify a given class within the Plugin server.

## Sample Uses of Plug Ins

Here are some common uses of Plugins:

<i>Use</i>	<i>Description</i>
Formula calculations	Replace or supplement formula calculations. A good example of this is tax calculations. Plugin writers can override all or portions of tax calculations.
Custom user interface actions	The following types of custom UI interfaces are supported: <ul style="list-style-type: none"> <li>▪ Side/top menu buttons</li> <li>▪ Text edit controls</li> <li>▪ Checkbox edit controls</li> <li>▪ ComboBox edit controls</li> </ul>
Custom attributes	Plugins usually affect the Retail Pro 9 data store via the defined business objects, but Plugins can store information in the application data store, in preferences, or in a table owned by and used exclusively by the Plugin.
Electronic funds transfer (EFT)	The tender Plugin interface implements Retail Pro, Inc.'s electronic funds transfer (EFT) link. Methods and events are tailored specifically to meet EFT requirements without precluding the use of those events for other purposes.



Validations	Plugins can be used to trap modifications to input fields and reject invalid values.
Building documents	Plugins can construct complex documents using the business rules defined in Retail Pro for each type of document.
Hardware support	Retail Pro uses the hardware Plugin interfaces to support a wide range of standard POS devices, and support for non-standard and custom devices can be created without modification to the Retail Pro executable.

### **Data Access by Plugins**

The rights granted on the database connection provided by the Plugin Adapter are highly restricted. Plugins can't affect data directly. They can read data using a SQL statement, but the only data they can affect without going through a business object are the Plugin tables.

### **Differences between Retail Pro 9 Plugins and Retail Pro 8 Plugins**

Here are the key differences in how Plugins work in Retail Pro 8 and Retail Pro 9:

- Retail Pro 8 uses the BTree Filer database system. Retail Pro 9 uses Oracle 11g or above.
- Retail Pro 8 uses Borland's BPL (.bpl) architecture for housing Plugin modules. The BPL architecture is version specific, so Retail Pro 8 Plugins can only be written using Delphi 5 (increasingly difficult to obtain) and Plugin functionality is restricted the functionality to functions supported by Delphi 5 and components available for that version.
- Retail Pro 9 uses Microsoft's Common Object Model (COM) as the method for locating and loading Plugin classes. The definition of the API is contained in a type library that is distributed to Plugin developers. Any language that supports COM and type libraries can use this library during development and compilation.
- Retail Pro 9 controls access to the database using *business objects*. Business objects codify the business rules surrounding a given logical entity, data storage, data relationships, data movement, user interface, and validations. The new Plugin architecture gives access to those business objects and even provides limited direct access to the underlying Oracle database.

### **Compatibility with Retail Pro8 Plugins**

Retail Pro 8, release 22 and above, had a completely different Plugin interface that used Delphi packages (BPLs) as the implementation mechanism for housing Plugin classes. The new Plugin API has nothing in common with the old API outside of coincidental similarities in method signatures, and the two cannot be used together.

Projects to move Plugins from Retail Pro 8 to 9 should be thought of as rewrites. If you're moving your development environment from Delphi 5 to a later version of Delphi, there's always the chance that you'll be able to reuse some of your code, but the architecture of Retail Pro and the interface into it is radically different, and you should approach the project as a port rather than a simple upgrade.

# Plugin Components and Life Cycle

To create a successful Plugin, it is important to understand:

- The key components of a Plugin
- How those components interact during the life cycle of the Plugin

## Key Components of a Plugin

The key components of a Plugin server are:

- The manifest file. This tells the Plugin manager how to locate the class that implements the IDiscover interface.
- The IDiscover interface. This tells the Plugin manager what classes are available in the COM server.
- The IConfigure interface. Implemented by one of the classes in the COM Plugin server and referenced in the manifest, this class tells the Plugin manager what classes are available in the server.
- The IPluginAdapter interface. A Plugin communicates with Retail Pro via its adapter using this interface. In some cases, as with the IPOSSessionAdapter, this adapter instance is specialized to handle specific Plugin requirements.
- The Plugin classes. The heart of the Plugin architecture.

## Plugin Life Cycle

*Note:* The life cycle of hardware and UI Plugins differs due to the nature of their use by the application.

The table describes Retail Pro's machinations, solely for the purpose of providing an understanding of what's going on behind the scenes within Retail Pro. For the most part, the Plugin writer doesn't need to worry about how it happens, just that the Plugin gets loaded and signaled when events it needs to handle occur.

<i>Steps</i>	<i>Notes/Comments</i>
Retail Pro is launched.	
Plugin manager looks in the <b>\RetailPro9\Plugins</b> directory for manifest (.mnf) file(s).	The discovery object for each server is expected to be represented by a single line in a manifest. <i>Reference:</i> See Manifest Files

<i>Steps</i>	<i>Notes/Comments</i>
The Plugin manager uses the server and discovery class name to request the class ID from Windows for that class. The retrieved class ID is then used to instantiate the discovery class.	<p>The Plugin manager obtains the class ID (<b>CLSID</b>) for the discovery class from Windows using the system call ProgIDtoCLSID. Using the CLSID, Retail Pro® 9 creates an instance of the class using the function CreateComObject. Then the manager calls IDiscover.PluginGUIDs to obtain a variant array of class names that the server provides. Each of the classes is loaded and its QueryInterface method called to determine what Plugin class it implements. Depending on the interface supported, the instance is queried for information regarding execution context.</p> <p>If you used regsvr32 is used to register the COM class, information about that COM class is saved in the registry under HKEY_CLASSES_ROOT. The retrieved class ID is then used to instantiate the discovery class.</p>
	<p>The Plugin manager obtains a list of the class names along with the following information from each class:</p> <ul style="list-style-type: none"> <li>▪ Interface supported (IAttributeValidation, ISideButton, etc.)</li> </ul> <p>And depending on the Plugin class:</p> <ul style="list-style-type: none"> <li>▪ Menu affected</li> <li>▪ Caption &amp; Picture file name (side menu buttons only)</li> <li>▪ Field type, label, and description (custom fields only)</li> </ul>
The supported API version is requested.	The Plugin's API version is checked against the application's API version to see if the Plugin can be supported. If not, the COM server's discovery object is freed and no further action is taken for Plugins in that COM server module.
Once the Plugin manager has a record of which Plugins are available and when they should be created, Retail Pro start-up continues.	In future versions of the PIAPI, there may be support for legacy 9.x Plugins, but backwards compatibility is not supported at this time.

Steps	Notes/Comments
The Plugin is instantiated.	<p>Plugins are instantiated, triggered, and freed (destroyed) based on the interface they implement and the context they specified during discovery.</p> <p><i>User-Interface Plugins</i></p> <p>User interface Plugins are instantiated when the business object they depend upon is created. The business object notifies the Plugin manager that it has been created via the <code>BOCreated</code> method. This method runs through each Plugin to see if the Plugin needs to be attached to the business object. If it does, a <i>business object wrapper</i> is first created. The business object wrapper acts as an intermediary between the business object and the Plugin adapter. This wrapper implements any additional restrictions and validations, such as further restricting access to attributes, perhaps allowing the Plugin to see most of the attributes but only allowing modifications to certain specific attributes.</p> <p>Then a <i>Plugin adapter</i> is created. The business object wrapper maintains a list of Plugin adapters as they are created.</p> <p>The Plugin adapter is given the Plugin's GUID, and calls the COM class factory to instantiate the Plugin.</p> <p>When all the Plugins are instantiated for the business object, the Plugin manager calls each adapter to run its Plugin's <code>Prepare</code> method. Running the <code>Prepare</code> method gives the Plugins a chance to initialize themselves. This also gives the Plugin adapter a chance to create and register the notifier objects required to receive messages from the business object and trigger the Plugin.</p> <p><i>Hardware Plugins</i></p> <p>Hardware Plugins are instantiated when the <code>POSStation</code> object is created. The <code>POSStation</code> is created whenever a POS document entry frame is brought up, including receipts, SOs, and POs. The <code>POSStation</code> object handles interaction between Retail Pro and hardware Plugins. The <code>POSStation</code> object functions as a globally visible adapter with properties and methods for accessing each of the various hardware devices Retail Pro supports. The <code>Prepare</code> method on these hardware Plugins should initialize the hardware device and prepare it for use.</p>

Steps	Notes/Comments
The Plugin receives a trigger or notification.	<p>User interface Plugin adapters receive notifications from business objects via the BNotify mechanism. Various points in the Retail Pro® application can call BNotify and pass a TBNotification descendant object to all notifier objects that are registered with that business object. When an adapter sees an event it needs to respond to, it calls the appropriate method on its Plugin. In the case of most of the Plugins, this is the HandleEvent method. If an event has multiple trigger types, it will likely have other methods for handling those calls. For example, classes that implement ITenderPlugin class must provide StartTransaction, AddTender, RemoveTender, Settle, and other methods that are called depending on what's required.</p> <p>Hardware interface Plugin adapters are, in some cases, not triggered by the application, but instead generate events that the application responds to by pulling the Plugin's properties to obtain the information the hardware device is providing.</p>
When a user interface Plugin has been triggered, it has access to the business object it is connected to via methods on the adapter. The Plugin can also request access to referenced business objects (business objects used to pull related information from other datasets) and request that new business objects be created.	<p>All business objects are accessed via <i>handles</i>, and attributes on those business objects are accessed by name. Simple preference settings are available by ID using the enumerated type BOPreference.</p> <p>Direct access to Retail Pro data is available by submitting SQL statements. Result sets are returned in variant arrays.</p> <p>If a Plugin needs to persist information, it can either do so using its own data storage mechanism or by submitting SQL statements that make calls to predefined stored procedures that give the Plugin access to a set of tables in the Retail Pro tablespace.</p> <p>Business object handles can be cloned, enabling Plugins to communicate directly with each other.</p>
Destruction occurs.	<p>When a business object is going out of scope, it calls the Plugin adapter's CleanUp method, which in turn calls the Plugin's CleanUp method, then calls the CleanUp method for any business objects that were created by the Plugin. This gives the Plugin the chance to free any allocated memory prior to being freed.</p>

## Business Objects

Retail Pro 9 uses business objects to codify the business rules surrounding a given logical entity. These business rules govern data storage, data relationships, data movement, user interface, and validations.

### *Retail Pro Business Object Creation*

When a business object is created, the Plugin manager's `BOCreated` method is called in the `TRTIAbstractBO` constructor, passing the new business object as the sole parameter. When the `BOCreated` method is called, the following steps are performed:

1. The Plugin manager creates a business object wrapper based on contextual information. If no contextual wrapper exists, the base wrapper, `TRTIAbstractBOW` is instantiated. The business object reference is placed in the business object wrapper's `WrappedBO` property and the business object wrapper reference is placed in the business object's `Wrapper` property.
2. The Plugin manager runs through its list of Plugin definitions for Plugins that need to be connected with this business object.
3. The Plugin manager then runs through its list of Plugin definitions and create adapters for the Plugins, adding each adapter to the business object wrapper's adapter list. The adapter's `BOW` property is populated with a reference to its business object wrapper.

When all the adapters have been created:

1. Each adapter's **LoadPlugin** method is called. The adapter uses COM to request an instance of its Plugin.
2. When all the Plugins have been loaded, each adapter calls its Plugin's `Prepare` method to allow for communication between the Plugins. This separation allows the Plugins to intercommunicate knowing that all the Plugins have been loaded.
3. The adapter then creates the necessary notifications to talk to its business object wrapper. Since adapters are specialized to work with individual Plugin interfaces, the adapter instantiates those notifiers that are pertinent to the interface it supports.
4. Depending on the interface supported, instances of `TBONotifyEvent` is created by the adapter and registered with the business object wrapper. The business object wrapper in turn registers a new `TBONotifyEvent` with its wrapped business object.
5. The reference to the business object is passed back to the calling routine.

### *Plugin Creation of New Business Objects*

It's possible for a group of related Plugins to communicate with each other. If such communication is needed, the Plugins could use shared memory, named pipes, or any other method to locate each other in memory and establish communications.

The Plugin wrapper maintains a list of business object handles and references that have either been opened via that adapter or cloned with that adapter by its Plugin. The handle is passed on to the Plugin. From that point, the Plugin can reference that business object by passing the handle to the adapter as a parameter to the adapter's methods.

If, during inter-Plugin communication, a Plugin wishes to pass a business object handle to another Plugin, the originating Plugin must clone the business object handle by calling its adapter. That adapter queries its business object wrapper for a reference to the child business object. Once that reference has been obtained, the adapter creates a new handle.

When a Plugin adapter calls the Plugin manager's CreateBOByXXX, it creates the business object the same as Retail Pro 9 would. As described above, the return value is a reference to the business object. The following steps are followed:

1. The Plugin manager pulls the new business object's business object wrapper property value and pass it to the requesting Plugin adapter's business object wrapper as a child business object wrapper.
2. The parent business object wrapper adds the child business object wrapper reference to its list of child business object wrappers.
3. The Plugin manager then passes back the child business object wrapper reference to the requesting adapter.
4. The requesting adapter adds the business object wrapper reference to its list of business object wrappers and pass back the index in that list of the business object wrapper to its Plugin as the "handle" for accessing that business object wrapper.

## **Sharing Child Business Objects**

### **Handles**

To ensure that clean-up of child business objects occurs correctly, all business objects are referenced by their handle. This handle is passed to the Plugin adapter as part of every method call. A handle of zero indicates the parent business object. If an adapter passes a handle to another Plugin, that Plugin must register the handle with its adapter. That adapter must request a business object reference from its parent business object.

### **Cloning Handles**

The series of steps involved in sharing a business object wrapper is as follows:

1. The source Plugin passes the target Plugin the source handle and a reference to the source Plugin adapter. This allows cloning of any business object wrapper handle that the source Plugin has knowledge of. The source adapter is passed because it has the business object wrapper reference and can tie it to the handle.
2. The target Plugin calls the target adapter's CloneHandle method, passing it the source handle and source adapter reference.
3. The target Plugin adapter calls the source Plugin adapter and requests the business object wrapper reference for that handle.

4. Once the target has the source business object wrapper reference, it adds that to its list of business object wrappers and passes a handle (the list index) back to the target Plugin. (Note that this increments the business object wrapper's reference count.)

At this point the target adapter has a reference to the source business object wrapper and the target Plugin has a handle with which to reference the source business object wrapper. The target Plugin must NIL its reference to the source Plugin adapter immediately. If the reference is not NILEd, the source adapter is held in memory during destruction and this will lead to a memory leak.

### ***Business Object Destruction***

When a business object destructs, there must be a clean-up of the children business object and adapters. The clean-up process is as follows:

1. The root business object wrapper (the business object wrapper with no parent business object wrapper) increments the reference count to each of its child business object wrappers to hold them in memory until all the Plugin adapters are gone.
2. The root business object wrapper then instructs each of its child business object wrapper to Cleanup. The Cleanup call is recursive in that each child business object wrappers calls each of its child business object wrappers to clean up.
3. The clean-up method calls each of the business object wrapper's Plugin adapters to clean up.
4. The adapters call their Plugin's CleanUp methods. The Plugins must NIL every reference they have to business objects and adapters and de-allocate all run-time allocated memory structures. When the clean-up call returns, it is assumed that all Plugins are effectively turned off.
5. The root business object wrapper then instructs each of its child business object wrappers to destruct. This call is recursive.
6. After all of a business object wrapper's children business object wrappers are destructed, it destructs each of its Plugin adapters.
7. Each Plugin adapter NILs its Plugin reference and destructs.
8. Once all the business object wrapper's adapters are freed, it destructs.

This process ensures that deallocation happens in the reverse order of allocation, guaranteeing (as much as possible) that no bad pointers occur and that nothing is held in memory beyond the end of its expected life cycle.



## Business Object Wrappers

Business object wrappers are used internally in Retail Pro to safely access business objects. The Plugin adapter, which each Plugin uses to communicate with Retail Pro, maintains a connection to a parent business object and a list of any child business objects the Plugin has requested. The business object wrapper's primary function is as an extension of the business object, limiting interaction with the business object to those functions that can be safely performed by code that is not part of the Retail Pro application core.

Specifically, business object wrappers:

- Provide access to preference settings as appropriate
- Perform any necessary additional security checks
- Provide read access to attribute values of business objects
- Modify access to particular attribute values of business objects

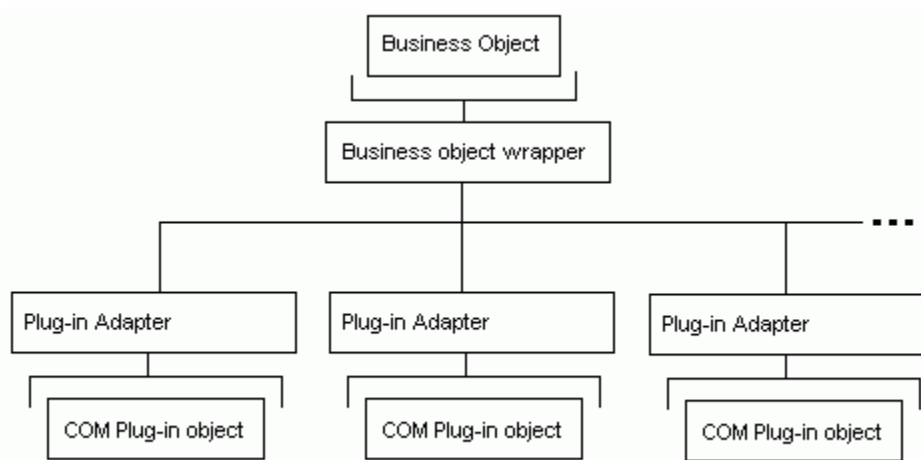
*Note:* For performance reasons, business objects do not always retrieve values of all attributes from the database. The list of attributes to be retrieved is defined by layouts and business logic. In some cases, a Plugin may need to change some of the Retail Pro 9 preference values or modify the interface used to collect those values to perform additional validations. An example is the Global Store Code. A Plugin that relies on this field, which is not currently a required field, may wish to require the user to supply this information and to format it in a certain way.

### *TRTIBOWrapper*

Internally in Retail Pro, the root business object wrapper is TRTIBOWrapper, which descends from TRTIAbstractBO with a single additional property, WrappedBO, implementing all its methods as pass-thru methods to WrappedBO's methods and properties of the same name. Within Retail Pro, this class is subclassed to either extend or limit the generic functions on a case-by-case basis.

Descendants of TRTIBOWrapper can restrict access to specific methods where appropriate, and there can be multiple wrappers defined for a given business object, each instantiated depending on the application context. There can be only one business object wrapper instantiated for a business object in any single context.

The business object wrapper is instantiated by the Plugin manager during the call from a new business object to the Plugin manager (BOCreated method). One business object wrapper is instantiated and shared by each Plugin adapter associated with that business object. The business object wrapper is a child component of the business object, and Plugin adapters are created as needed as child components of the business object wrapper. When the business object destructs, the wrappers and adapters are destroyed as well.

**Run-time configuration of a business object with Plugins attached:****Communication between Plugins**

The business object wrapper maintains a list of dependent Plugin adapters. The adapters can access this list to construct a list of COM interfaces to other Plugins. This list can be passed to a Plugin upon request. The Plugin can then cycle through the COM references and communicate directly with the other Plugins. Because the Plugins then have references to each other and possibly to each other's adapters, business objects, etc., all Plugins must implement a "CleanUp" method so that these references can be set to NIL when the business object goes out of scope. Otherwise, by referencing each other, they prevent their reference counts from reaching zero, at which time they should free themselves. This would lead to a serious memory leak.

**TenderBO Wrapper**

This section explains the TenderBO Wrapper, which is a business object wrapper used for special tender situations.

**Background**

Retail Pro's tenders are typically handled by ITenderDialoguePlugin and ITenderPlugin. ITenderDialoguePlugin is used to replace one of RetailPro's internal tender logins. ITenderPlugin notifies the PI API that the tender screen is entered or tender info changed.

TenderBO, on the other hand, is used in the following primary situations:

- A developer needs to create and populate an Invoice and its' tenders then save this data to the RetailPro 9 database. As an example; porting legacy data into RetailPro 9.
- A developer wants to override the RetailPro 9 Invoice process. In this case, the developer would write the Invoice and tenders screen completely and then store the subsequent data in the retailPro9 Database.

These are both situations in which a developer needs to override RetailPro's standard tenders due to the unique requirements of the developer's problem domain. For this reason, normal validation requirements and business rules are bypassed and are left to the Plugin developer to design and implement.

### **Available Operations**

The developer will be able to perform the follow operations: Open, Close, Get, Set, Edit, Post, Delete and navigate a Receipts Tender data. Please note that all validation and/or business rules associated with the TenderBO are the responsibly of the Plugin developer.

### **Methods**

The methods available to the TenderBO are enumerated below. They will have the same parameters and return the same types as provided in IPluginAdapter.

BOSetAttributeValueByName	BOClose
BOGetAttributeValueByName	BOEdit
BOSetAttributeValues	BOInsert
BOGetAttributeValues	BODelete
BOGetState	BOPost
BOSetActive	BOCancel
BOGetActive	BOFirst
BOGetAttributeNameList	BOLast
BOOpen	BONext
BOClear	BOPrior
BOF	EOF

### **Parent Business Object**

The TenderBO is accessed as a child of the Invoice Business Object.

### **Pseudo Code Examples**

Below are pseudo code examples of how a TenderBO would be utilized:

Assuming that you already have a handle to the InvoiceBO (0), you would proceed to access the TenderBO and add a tender.

```
TenderHandle := Adapter.GetReferenceBOForAttribute( 0, 'Tenders' )
```

```
Adapter.BOOpen( TenderHandle )
```

```
Adapter.BOInsert( TenderHandle )
```

```
Adapter.BOSetAttributeValueByName( TenderHandle, 'TenderType', ttCash )
```

```
Adapter.BOSetAttributeValueByName( TenderHandle, 'GIVEN" , 1000.00 )
```

```
Adapter.BOSetAttributeValueByName( TenderHandle, 'TAKEN" , 500.00 )
```

## TenderTypes

TenderBO supports the following RetailPro9 Tender Types.

ttCash	*ttDeposit
ttCheck	ttPayments
ttCreditCard	ttGiftCertificate
ttCOD	ttGiftCard
Charges	ttDebitCard
ttStoreCredit	ttForeignCurrency
ttRESERVED	ttTravelerCheck

\* ttDeposit is only supported in SalesOrder and SalesOrder is not supported in this implementation of TenderBOWrapper.

## Attributes

All of the TenderBO attributes are listed below. It should be noted that not all TenderTypes are created equal and as such not all attributes are implemented for each and every TenderType. See the **Tender Attribute cross Reference Table** following the **Attribute definition Table** for details.

### Attribute Definition Table

Name	Type	Comments
AMT	NUMBER(16,4)	
AUTH	NVARCHAR2(26)	
AVS_CODE	VARCHAR2(4)	AVS response code from processor
CHARGE_DISC_DAYS	NUMBER(10)	
CHARGE_DISC_PERC	NUMBER(16,4)	
CHARGE_NET_DAYS	NUMBER(10)	
CHK_COMPANY	NVARCHAR2(25)	
CHK_DL	NVARCHAR2(30)	
CHK_DL_EXP_DATE	DATE	Concatenated MMDDYYYY
CHK_DOB_DATE	DATE	Concatenated MMDDYYYY
CHK_FIRST_NAME	NVARCHAR2(30)	
CHK_HOME_PHONE	NVARCHAR2(11)	
CHK_LAST_NAME	NVARCHAR2(30)	
CHK_STATE_CODE	NVARCHAR2(2)	

<i>Name</i>	<i>Type</i>	<i>Comments</i>
CHK_TYPE	NUMBER(1)	0=Personal 1=Business
CHK_WORK_PHONE	NVARCHAR2(11)	
CRD_CONTR_NO	NVARCHAR2(19)	
CRD_EXP_MONTH	NUMBER(5)	MM
CRD_EXP_YEAR	NUMBER(10)	YY
CRD_NORMAL_SALE	NUMBER(1)	
CRD_PRESENT	NUMBER(1)	
CRD_PROC_FEE	NUMBER(16,4)	
CRD_TYPE	NUMBER(5)	
CRD_ZIP	NVARCHAR2(10)	
CURRENCY_ID	NUMBER(5)	
CURRENCY_NAME	NVARCHAR2(30)	Used for historical purposes only. Records original currency name of the currency_id at the time the record was created.
DOC_NO	NVARCHAR2(30)	
EFTDATA0	CLOB	Additional data that is needed for printing on the invoice
EFTDATA1	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA2	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA3	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA4	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA5	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA6	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA7	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA8	NVARCHAR2(32)	Additional data that is needed for printing on the invoice
EFTDATA9	NVARCHAR2(32)	Additional data that is needed for printing on the invoice

<i>Name</i>	<i>Type</i>	<i>Comments</i>
Name	Type	Comments
FAILURE_MSG	VARCHAR2(64)	The error message returned for failed transactions
GFT_CRD_BALANCE	NUMBER(16,4)	
GFT_CRD_INT_REF_NO	NVARCHAR2(11)	
GFT_CRD_TRACE_NO	NVARCHAR2(11)	
GIVEN	NUMBER(16,4)	
INVC_SID	NUMBER(19)	supplied by InvoiceBO
L2_RESULT_CODE	NCHAR(1)	Result code for commercial card processing
MANUAL_NAME	NVARCHAR2(80)	
MANUAL_REMARK	NVARCHAR2(40)	
MATCHED	NUMBER(1)	Used by the XZOut application to mark an invoice tender record as being matched to tender in the workstation/drawer being counted.
ORIG_CRD_NAME	NVARCHAR2(6)	Credit card name at the time the transaction was tendered
PMT_DATE	DATE	Concatenated MMDDYYYY
PMT_REMARK	NVARCHAR2(21)	
PROC_DATE	DATE	Concatenated MMDDYYYY, Represents when the processing of the EFT tender occurred
REFERENCE	NVARCHAR2(8)	
SIGNATURE_MAP	BLOB	Stores point array information collected by signature capture device. Stored as a string.
SIGNATURE_MAP_CLOB	CLOB	Stores point array information collected by signature capture device. Stored as a string.
TAKEN	NUMBER(16,4)	
TENDER_NO	NUMBER(5)	In case of Cash - Currency_Id, In case of Credit Card - Credit Card sequential number, In case of Payments - Payment_No, Otherwise - just 1

Name	Type	Comments
TENDER_STATE	NUMBER(5)	Contains the current state of each tender in relation to EFT Res. It will be populated with one of the following values. 0 = Tender processed online normal 1 = Manual offline pending processing 2 = Auto offline pending processing 3 = Performing processing 4 = Processed success 5 = Processed failure 6 = Voice auth. offline pending processing
TENDER_NAME	NVARCHAR(20)	<b>READ ONLY</b> , an attempt to set value returns peUnableToSetAttributeValue.
TENDER_TYPE	NUMBER(5)	REQUIRED
TRANSACTION_ID	NVARCHAR2(26)	Stores the gateways record id for the EFT transaction.

### **Tender Attribute cross Reference Table**

The table below describes the tender types by which each attribute can be accessed.

Tender Type	Attributes	Required	Default
<b>Cash</b>			
ttCash	TENDER_TYPE	*	NA
ttCash	TENDER_NAME		NA
ttCash	TAKEN	*	0
ttCash	GIVEN	*	0
ttCash	MANUAL_REMARK		""
<b>Check</b>			
ttCheck	TENDER_TYPE	*	NA
ttCheck	TENDER_NAME		NA
ttCheck	AMT	*	0
ttCheck	MANUAL_REMARK		""
ttCheck	DOC_NO		""
ttCheck	AUTH		""
ttCheck	TRANSACTION_ID		0
ttCheck	CHK_TYPE		0
ttCheck	PROC_DATE		null

<i>Tender Type</i>	<i>Attributes</i>	<i>Required</i>	<i>Default</i>
ttCheck	FAILURE_MSG		""
ttCheck	TENDER_STATE		0
ttCheck	EFTDATA0		null
ttCheck	EFTDATA1		""
ttCheck	EFTDATA2		""
ttCheck	EFTDATA3		""
ttCheck	EFTDATA4		""
ttCheck	EFTDATA5		""
ttCheck	EFTDATA6		""
ttCheck	EFTDATA7		""
ttCheck	EFTDATA8		""
ttCheck	EFTDATA9		""
Credit Card			
ttCreditCard	TENDER_TYPE	*	NA
ttCreditCard	TENDER_NAME		NA
ttCreditCard	AMT	*	0
ttCreditCard	DOC_NO		""
ttCreditCard	AUTH		""
ttCreditCard	CRD_EXP_MONTH		0
ttCreditCard	CRD_EXP_YEAR		0
ttCreditCard	CRD_TYPE		0
ttCreditCard	TRANSACTION_ID		0
ttCreditCard	CRD_PRESENT		0
ttCreditCard	AVS_CODE		0
ttCreditCard	SIGNATURE_MAP		null
ttCreditCard	MANUAL_REMARK		""
ttCreditCard	PROC_DATE		null
ttCreditCard	FAILURE_MSG		""
ttCreditCard	TENDER_STATE		0
ttCreditCard	EFTDATA0		null
ttCreditCard	EFTDATA1		""
ttCreditCard	EFTDATA2		""
ttCreditCard	EFTDATA3		""
ttCreditCard	EFTDATA4		""
ttCreditCard	EFTDATA5		""
ttCreditCard	EFTDATA6		""



<i>Tender Type</i>	<i>Attributes</i>	<i>Required</i>	<i>Default</i>
ttCreditCard	EFTDATA7		""
ttCreditCard	EFTDATA8		""
ttCreditCard	EFTDATA9		""
COD			
ttCOD	TENDER_TYPE	*	NA
ttCOD	TENDER_NAME		NA
ttCOD	AMT	*	0
ttCOD	MANUAL_REMARK		""
Charge			
ttCharge	TENDER_TYPE	*	NA
ttCharge	TENDER_NAME		NA
ttCharge	AMT	*	0
ttCharge	CHARGE_DISC_PERC		0
ttCharge	CHARGE_DISC_DAYS		0
ttCharge	CHARGE_NET_DAYS		0
ttCharge	MANUAL_REMARK		""
Store Credit			
ttStoreCredit	TENDER_TYPE	*	NA
ttStoreCredit	TENDER_NAME		NA
ttStoreCredit	AMT	*	0
ttStoreCredit	DOC_NO		""
ttStoreCredit	AUTH		""
ttStoreCredit	MANUAL_REMARK		""
Deposit			
ttDeposit	TENDER_TYPE	*	NA
ttDeposit	TENDER_NAME		NA
ttDeposit	Not supported		0
ttDeposit	Not supported		0
Payments			
ttPayments	TENDER_TYPE	*	NA
ttPayments	TENDER_NAME		NA
ttPayments	AMT	*	0
ttPayments	PMT_DATE		null
ttPayments	PMT_REMARK		""
Gift Certificate			
ttGiftCertificate	TENDER_TYPE	*	NA

<i>Tender Type</i>	<i>Attributes</i>	<i>Required</i>	<i>Default</i>
ttGiftCertificate	TENDER_NAME		NA
ttGiftCertificate	AMT	*	0
ttGiftCertificate	DOC_NO		""
ttGiftCertificate	AUTH		""
ttGiftCertificate	MANUAL_REMARK		""
Gift Card			
ttGiftCard	TENDER_TYPE	*	NA
ttGiftCard	TENDER_NAME		NA
ttGiftCard	AMT	*	0
ttGiftCard	DOC_NO		""
ttGiftCard	AUTH		""
ttGiftCard	CRD_EXP_MONTH		0
ttGiftCard	CRD_EXP_YEAR		0
ttGiftCard	TRANSACTION_ID		0
ttGiftCard	CRD_PRESENT		0
ttGiftCard	GFT_CRD_BALANCE		0
ttGiftCard	MANUAL_REMARK		""
ttGiftCard	PROC_DATE		null
ttGiftCard	FAILURE_MSG		""
ttGiftCard	TENDER_STATE		0
ttGiftCard	EFTDATA0		null
ttGiftCard	EFTDATA1		""
ttGiftCard	EFTDATA2		""
ttGiftCard	EFTDATA3		""
ttGiftCard	EFTDATA4		""
ttGiftCard	EFTDATA5		""
ttGiftCard	EFTDATA6		""
ttGiftCard	EFTDATA7		""
ttGiftCard	EFTDATA8		""
ttGiftCard	EFTDATA9		""
Debit Card			
ttDebitCard	TENDER_TYPE	*	NA
ttDebitCard	TENDER_NAME		NA
ttDebitCard	AMT	*	0
ttDebitCard	DOC_NO		""
ttDebitCard	AUTH		""

<i>Tender Type</i>	<i>Attributes</i>	<i>Required</i>	<i>Default</i>
ttDebitCard	CRD_EXP_MONTH		0
ttDebitCard	CRD_EXP_YEAR		0
ttDebitCard	TRANSACTION_ID		0
ttDebitCard	CRD_PRESENT		0
ttDebitCard	GFT_CRD_BALANCE		0
ttDebitCard	MANUAL_REMARK		""
ttDebitCard	PROC_DATE		null
ttDebitCard	FAILURE_MSG		""
ttDebitCard	TENDER_STATE		0
ttDebitCard	EFTDATA0		null
ttDebitCard	EFTDATA1		""
ttDebitCard	EFTDATA2		""
ttDebitCard	EFTDATA3		""
ttDebitCard	EFTDATA4		""
ttDebitCard	EFTDATA5		""
ttDebitCard	EFTDATA6		""
ttDebitCard	EFTDATA7		""
ttDebitCard	EFTDATA8		""
ttDebitCard	EFTDATA9		""
Foreign Currency			
ttForeignCurrency	TENDER_TYPE	*	NA
ttForeignCurrency	TENDER_NAME		NA
ttForeignCurrency	TAKEN	*	0
ttForeignCurrency	GIVEN	*	0
ttForeignCurrency	AMT	*	0
ttForeignCurrency	CURRENCY_ID	*	0
ttForeignCurrency	MANUAL_REMARK		""
Traveler's Check			
ttTravelerCheck	TENDER_TYPE	*	NA
ttTravelerCheck	TENDER_NAME		NA
ttTravelerCheck	AMT	*	0
ttTravelerCheck	DOC_NO		""
ttTravelerCheck	MANUAL_REMARK		""
ForeignCheck			
ttForeignCheck	TENDER_TYPE	*	NA
ttForeignCheck	TENDER_NAME		NA

<i>Tender Type</i>	<i>Attributes</i>	<i>Required</i>	<i>Default</i>
ttForeignCheck	CURRENCY_ID	*	0
ttForeignCheck	CURRENCY_NAME	*	""
ttForeignCheck	TAKEN	*	0
ttForeignCheck	GIVEN	*	0
ttForeignCheck	AMT	*	0
ttForeignCheck	DOC_NO		""
ttForeignCheck	AUTH		""
ttForeignCheck	MANUAL_REMARK		""

## Plugin Manager

The Plugin manager collects information about the Plugins available to the application, discovers what contexts the Plugin classes are intended to supplement, and acts as a class factory for business object wrappers and Plugin adapters.

When instantiating a business object wrapper, the Plugin manager is supplied with a constant that describes which business object to create.

When creating a Plugin adapter, the Plugin manager is supplied with the business object constant and a constant that describes the Plugin type. If a Plugin COM object has been registered for that type, the appropriate Plugin adapter is instantiated and the reference is passed back to the business object wrapper, which maintains a list of adapters.

### COM Server Object Initialization

Retail Pro loads each Plugin module as a COM server object.

The Plugin manager obtains the class ID (CLSID) for IDiscovery from Windows using the ProgIDtoCLSID call. Using the CLSID, Retail Pro creates an instance of the class using the function CreateComObject. Then the manager calls IDiscover.PluginGUIDs to obtain a variant array of class names that the server provides.

Each of the classes is loaded and its QueryInterface method called to determine what Plugin class it implements. Depending on the interface supported, the instance is queried for the following information:

- Interface supported (IAttributeValidation, ISideButton, etc.)
- Business objects affected
- Attributes affected
- Caption & Picture file name (side menu buttons only)
- Field type, label, and description (custom fields only)

The discovery information (except for the interface supported) is passed back to the Plugin manager in variant arrays of strings.

## Plugin Adapters

The Plugin adapter manages the transfer of signals and method calls between the Plugin and the business object wrapper.

The adapter is the Plugin's attachment to the application, enabling the Plugin to access business objects, business objects referenced by the business object's attributes, simple preference settings, and a SQL session for accessing the Retail Pro tablespace directly. In some cases, as with the IPOSStationAdapter, the Plugin adapter is specialized to handle specific Plugin requirements.

Plugin adapters are created, as needed, as child components of the business object wrapper. When the business object destructs, the adapters (and wrappers) are destroyed as well.

### *Business Object Wrapper Handles*

The Plugin makes calls to the business object wrapper via the plug-adapter using a business object wrapper *handle*. Handles are used because Plugins typically need to refer to three kinds of business objects:

- The parent business object wrapper (always handle 0 (zero))
- Child business objects created upon request from the Plugin.
- Business object wrapper references passed via inter-Plugin communication

*Note:* A child business object wrapper is one in which the Plugin has requested that a new business object be instantiated. Plugins from a single server might wish to share business object references. In this case, a mechanism exists for cloning handles to business objects.

### *Adapter Notifications*

Plugin adapters create whatever notifications they require between themselves and the business object wrapper and the business object wrapper creates the same notification between itself and its wrapped business object.

User interface Plugin adapters receive notifications from business objects via the BONotify mechanism. Retail Pro can call BONotify and pass a TBONotification descendant object to all notifier objects that are registered with that business object.

When an adapter sees an event it needs to respond to, it calls the appropriate method on its Plugin. For most Plugins, this is the HandleEvent method. If an event has multiple trigger types, it will likely have other methods for handling those calls. For instance, classes that implement the ITenderPlugin class must provide StartTransaction, AddTender, RemoveTender, Settle, and other methods that are called depending on what's required.

Hardware interface Plugin adapters are, in some cases, not triggered by the application, but instead generate events that the application responds to by pulling the Plugin's properties to obtain the information the hardware device is providing.

## Notifications/Events

The Retail Pro 9 business object model supports the use of notification classes. Notifications serve as a generic link between objects that must observe changes in each other's state.

Notifications are applied to:

- Data events
- Document events
- Layout events

### *Data Events*

Data events are handled using the notification model already in place for descendents of the TAbstractBO class. The TAbstractBO class supports the following data events:

- Before Insert
- After Insert
- Before Update
- After Update
- Before Cancel
- After Cancel
- Validate Attribute Value
- Attribute Value Changed

### *Document Events*

The Retail Pro 9 API implements events that are triggered when items are added to documents, like vouchers, slips, receipts etc., all descended from TDocumentBO (which replaces TDocument in Retail Pro 8). Several items can be added at once, and the changes are refreshed in the business object only after all the items have been added and all records have been retrieved from the database.

TDocltemBO supports the BeforeAddItem and AfterAddItem methods. Both of these methods are triggered once for each item added to a document.

## Business Object User Interface Events (BOUIEvents)

Business Object User Interface Events (BOUIEvents) enable developers to respond to Retail Pro UI events in order to facilitate a seamless work flow.

Most BOUIEvents are in the form of a message dialog, in which a message is displayed to the user and the user clicks Yes or No (or OK/Cancel). However, some BOUIEvents are more complex; therefore, Retail Pro has tried to provide a flexible mechanism that can respond to a variety of events.

BOUIEvents are independent of any other Plugin events that may be triggered. The UI events can be supported via the ISideButtonPlugin and IServerPlugin interfaces even if those Plugins will not be used for any other purpose.

**NOTE:** If there is more than one Plugin that implements a specific BOUIEvent, after the first Plugin has handled the specific event (returns AHandled = true), no other Plugins that also handle that specific event will be triggered.

**IMPORTANT!** This event method is only utilized by RetailPro if your Plugin specifies it supports this method. Your Plugin **must** specify it supports HandleBOUIEvent via the PluginCapability method. If you are using CustomPluginClasses, the default response for PluginCapability is FALSE. You will have to override this method in your Plugin and respond as follows:

To enable your Plugin to respond to these types of events, PluginCapability must return TRUE when the ACapability parameter equals sbcHandleBOUIEvent.

If PluginCapability returns FALSE when the ACapability parameter equals sbcHandleBOUIEvent, then RetailPro will not call the Plugin to respond to BOUIEvents.

### Interfaces that Support HandleBOUIEvent

Currently only **ISideButtonPlugin** and **IServerPlugin** interfaces implement HandleBOUIEvent. These interfaces can be used to intercept and handle the supported UI events that occur at any time.

### UI Events that HandleBOUIEvent supports

Below is a list of the current UI Events where HandleBOUIEvent is supported and the values passed in AEventType. NOTE: All AEventName parameters MUST be upper case.

Location of Event	AEventType	Notes
Multiple BOs	'MESSAGEDLG'	Event for handling the generic MESSAGEDLG dialog. Can occur in various places and will have various titles.
Multiple BOs	'CONFIRMPERMANENTDELETE'	Event for handling the Confirm Permanent Delete dialog. Can occur with multiple BO's.
InvoiceBO	'SELECTFEE'	Displayed when a fee must be selected on a receipt.

<i>Location of Event</i>	<i>AEventType</i>	<i>Notes</i>
CustomerBO	'DEFAULTCUSTOMERTYPE'	Displayed when creating a new customer and no default customer type has been set in preferences.
VoucherBO	'CONFIRMINVENTORYUPDATE'	Event for handling the Confirm Inventory Update dialog.
Vouchers	'HOLDBUTTONCLICK'	This event occurs before the HOLD button processing has occurred. (See UI Button Clicks section for more info)
Invoices	'HOLDBUTTONCLICK'	This event occurs before the HOLD button processing has occurred. (See UI Button Clicks section for more info)
Invoices	'ENTERTENDEREVENT'	This event occurs after the TENDER button has been pressed and all processing has occurred.
SalesOrder	'SALERECORDED' Or 'RETURNRECORDED'	<p>This event occurs at the end of the process of generating a receipt in the Sales Order area (when the Record Sale or Record Return button is clicked)</p> <p>Record Sale button - BOUIEventType = 'SALERECORDED'</p> <p>Record Return button (same button) - BOUIEventType = 'RETURNRECORDED'</p> <p>Parameters passed will be:</p> <p>Parameter 1 = SO SID</p> <p>Parameter 2 = Invoice SID of the new invoice receipt</p> <p>Note: This event is also called when a receipt is automatically generated and updated based on the preferences "Automatically Record Sale After Final Deposit" and "Require Record Sale before First Deposit."</p>





**Relevant methods:**

Get\_BOUIEventsSupported – the property method getter - indicates in a variant array the specific UI events handled by this Plugin

function Get\_BOUIEventsSupported: OleVariant; safecall;

HandleBOUIEvent – the event handler method for handling UI events

```
function HandleBOUIEvent( ABOHandle: Integer;
                        AEventType: PChar;
                        AParameters: OleVariant;
                        var AReturnValues: OleVariant;
                        var AHandled: WordBool): WordBool; safecall;
```

where:

ABOHandle = handle to the BO that the event was generated for

AEventType = the specific UI event that is being triggered (or handled)

AParameters = the parameters from the UI in a variant array

AReturnValues = the value the developer returns from the event handler – varies for each UI event

AHandled = was the event handled by this event handler - True or False

function result = should RetailPro continue processing this event – this should be the logical opposite of the AHandled value

Example implementation:

```
function TMyPlugin.PluginCapability( ACapability: Integer): WordBool;
begin
    Result := ACapability = sbcHandleBOUIEvent;
end;

function TMyPlugin.Get_BOUIEventsSupported: OleVariant;
var
begin
    result := VarArrayCreate( [ 1, 1 ], varOleStr );
    result[ 1 ] := 'CONFIRMINVENTORYUPDATE';
end;

function TMyPlugin.HandleBOUIEvent( ABOHandle: Integer;
AEventType: PAnsiChar;
AParameters: OleVariant;
var AReturnValues: OleVariant;
var AHandled: WordBool): WordBool;

var
    Successful: Boolean;
begin
    if AEventType = 'CONFIRMINVENTORYUPDATE' then
        begin
            // do some processing here and set the Successful variable

            If Successful then
                begin
                    AReturnValues = mrOk;
                    AHandled := true;
                    Result := false;
                end
            Else
                begin
                    AReturnValues = mrCancel;
```

```
AHandled := false;  
Result := true;  
end;  
end;
```

## UI Button Clicks

The HandleBOUIEvent event can listen and respond to user-interface button clicks.

In both the ISideButtonPlugin and IServerPlugin interfaces, when one of the implemented UI button clicks occurs, the HandleBOUIEvent will fire.

HandleBOUIEvent has many parameters, most of which are ignored with UI Button Clicks. The only parameters that HandleBOUIEvent uses upon a button click are the AEventType and ABOHandle parameters.

No other parameters are set and ALL return values are ignored. This is ONLY a button click event notification.

Because of the nature of the UI hooks, these UI Button Click events are not automatically available to every UI component. If you need an additional UI Button Click event, please let us know and we will consider implementing the additional event.

# Creating a Plugin in Delphi

## Setting Up Your Development Environment

To create Plugins for Retail Pro 9, you must have the following files installed in the \RetailPro9\ directory in your development environment:

<i>File</i>	<i>Notes</i>
RPRO9.exe	Version 9.2 (R5)
Plugins.tlb	Type library for the Plugin interface
MSXML.tlb	Microsoft XML type library

## Creating a Plugin - Basic Steps

**Important!** These instructions assume that you are using Delphi to create the COM server and Plugins.

### **Basic Steps to Create a Plugin:**

1. Import the Plugins type library (Plugins.tlb).
2. Create a new COM server object
3. Create a new type library and create CoClasses for Plugins.tlb.
4. Create the Plugin unit.
5. Create the Plugin class.
6. Create a discovery class.
7. Create instances of the class factory for each Plugin.
8. Export stored Procedures.
9. Compile your project and copy the resulting DLL to the RetailPro9\Plugins directory.
10. Register the COM Servers.
11. Create the manifest file and copy it to the ...\RetailPro9\Plugins folder.
12. Test the Plugin with Retail Pro 9.x.

## Importing the Plugins Type Library (Plugins.tlb)

You need to make Delphi aware of the Retail Pro Plugins type library so Delphi will know how to compile code that uses the type library. Do this by importing Plugins.tlb.

**Note that this will need to be done each time a new version of the library is distributed.**

1. Select Project | Import Type Library to bring up the import type library dialog.
2. Click Add, browse to the location where Plugins.tlb is located, and double click it.
3. Click Create unit. Note: uncheck the option "Generate Component Wrapper. Delphi imports the type library and generates a Plugins\_tlb.pas file to be included in the Uses clause of any unit that references it.
4. Select File | Close All. Proceed to Creating a New COM Server Object.

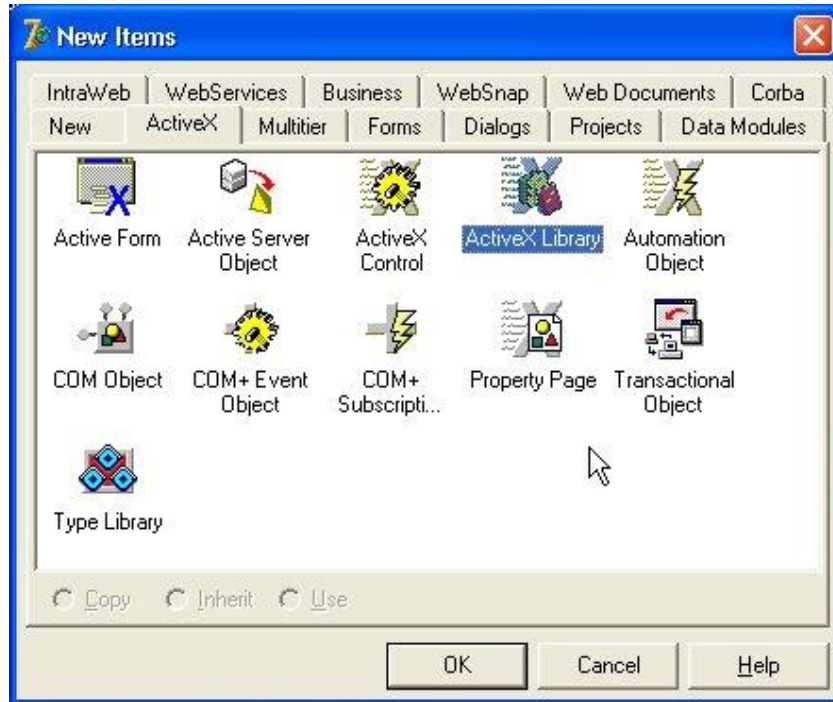
**Note:** If you inadvertently used the Install button on the Import Type Library dialog the first time you imported Plugins.tlb, Delphi will present a dialog that indicates certain interfaces are already installed when you attempt to create a unit for a new version of Plugins.tlb. Ignore this dialog and continue.

## Creating a New COM Server Object

**Important!** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

1. Select File | New | Other. The project templates window is displayed.

2. On the ActiveX tab, double click the ActiveX Library icon.



3. Press **Ctrl-S** and give your new empty COM server a name. Ours will be *CustomSideButtonTemplate*. The COM server you created is added to the list of COM servers.
4. Proceed to importing the Plugins creating a New Type Library.

## Creating a New Type Library

The type library that you create in this step is:

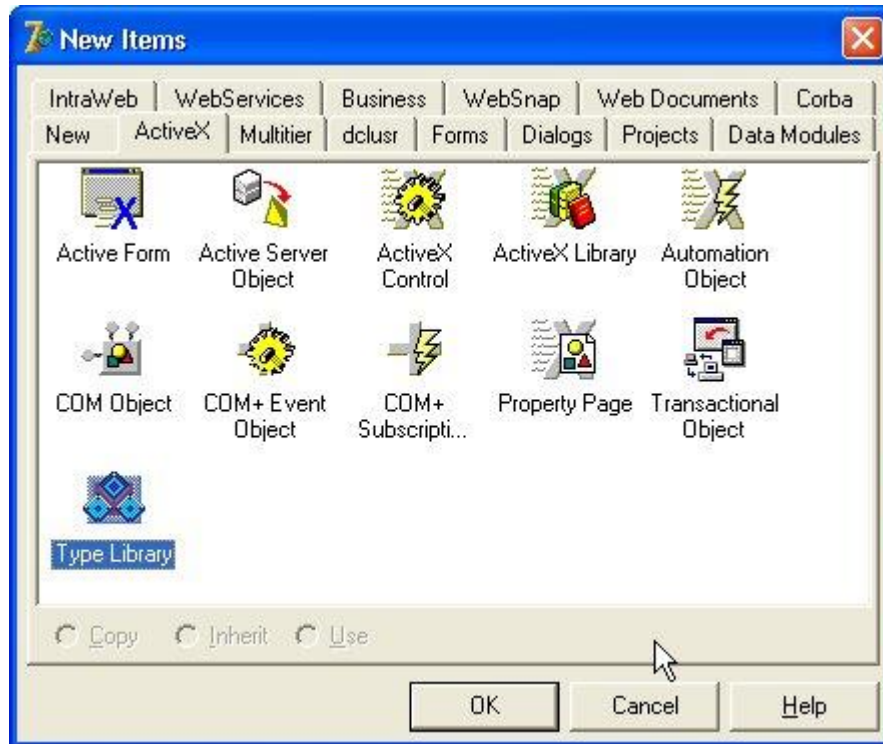
- The method for communicating the interface structures from Retail Pro to the Plugin
- The mechanism for defining the concrete implementation names back to Retail Pro® from the Plugin

**Important!** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

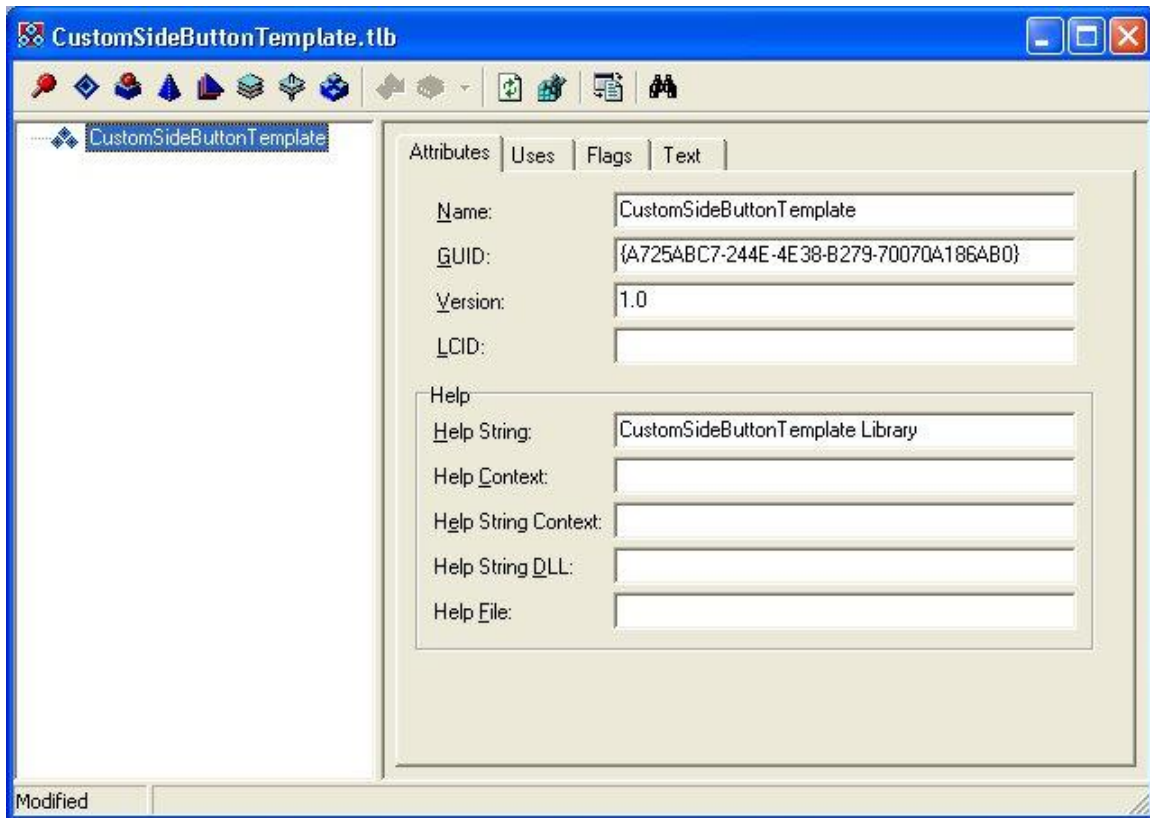
### **To create a new type library:**

1. Select File | New | Other to bring up the project templates window.

2. On the ActiveX tab, double click the Type Library icon.



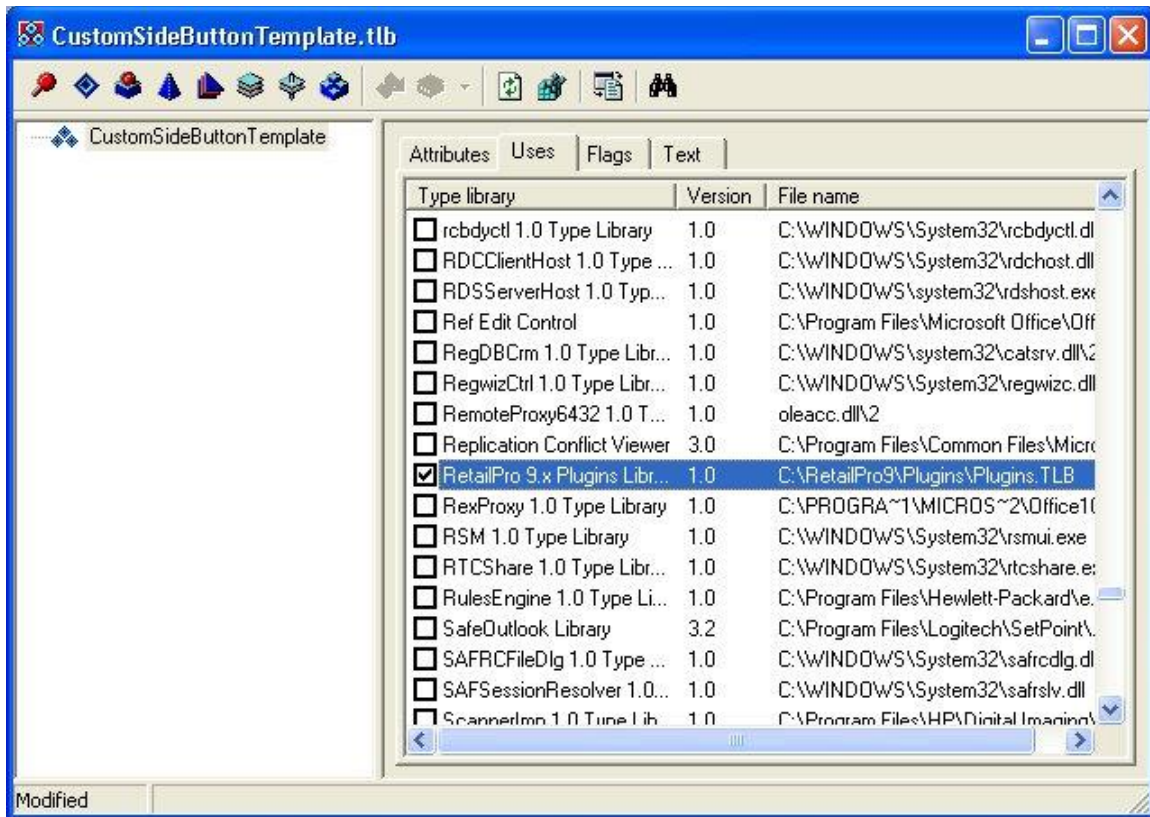
3. When the new type library comes up in the TLB editor, the Attributes page shows the type library's name, derived from the COM server name and a descriptive Help String.



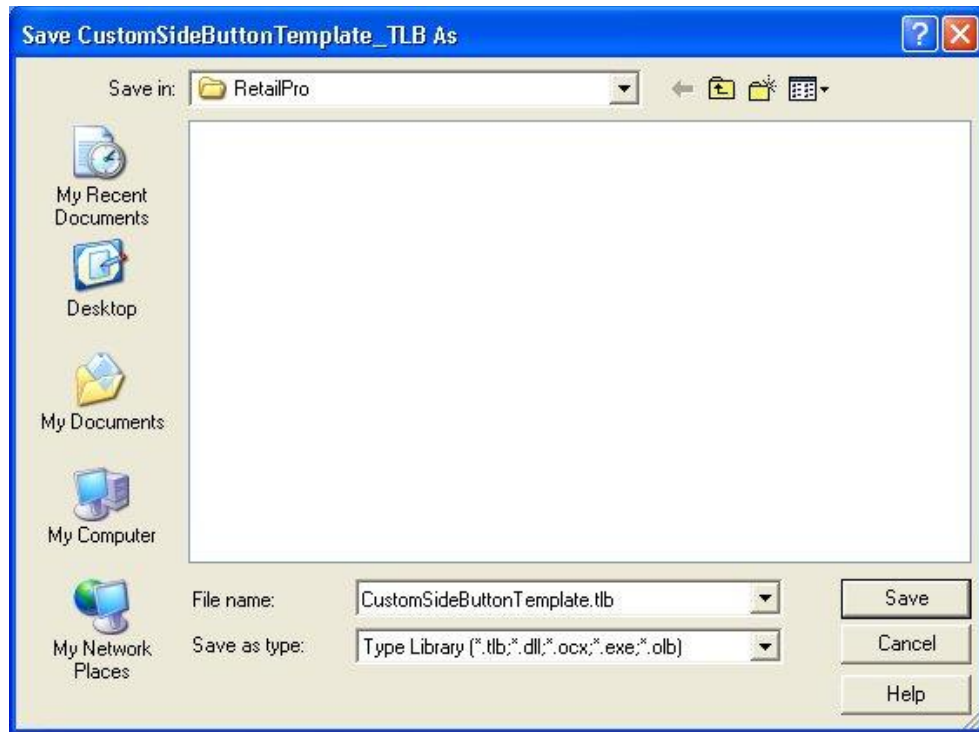
4. On the Uses page, right click on the grid and select Show All Type Libraries.



5. Select Retail Pro 9.x Plugins Library.



6. Press **Ctrl-S**, specify the library name, and click OK.

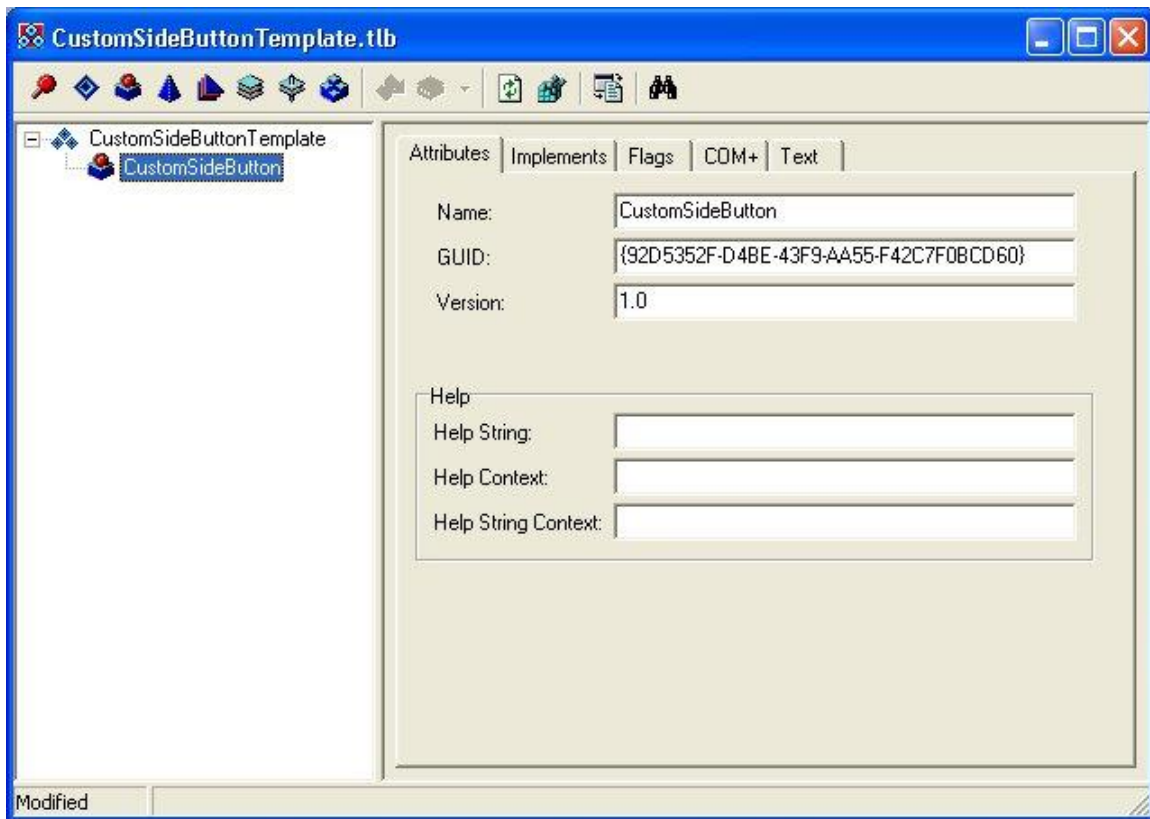


*Result:* You now have an empty type library that can reference interfaces in the Plugins Type Library (Plugins.tlb).

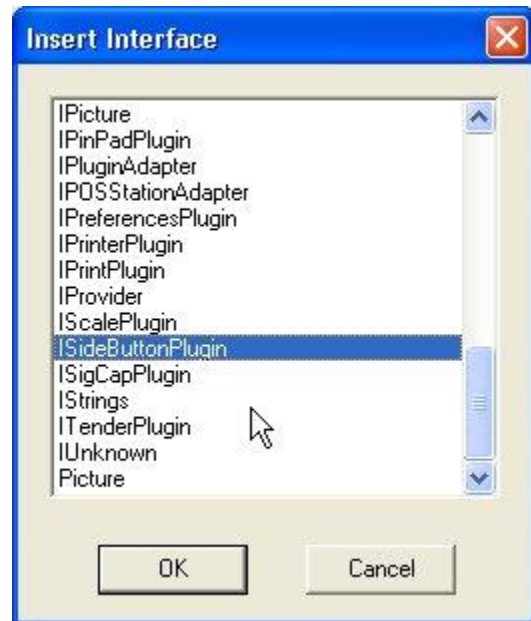
7. Next, you need to create CoClasses in the Plugins type library (see steps below). For each Plugin you wish to make visible to the Plugin manager, you'll need to make a CoClass that implements one of the concrete implementations from the Plugins type library.

### **To create CoClasses for Plugins.tlb:**

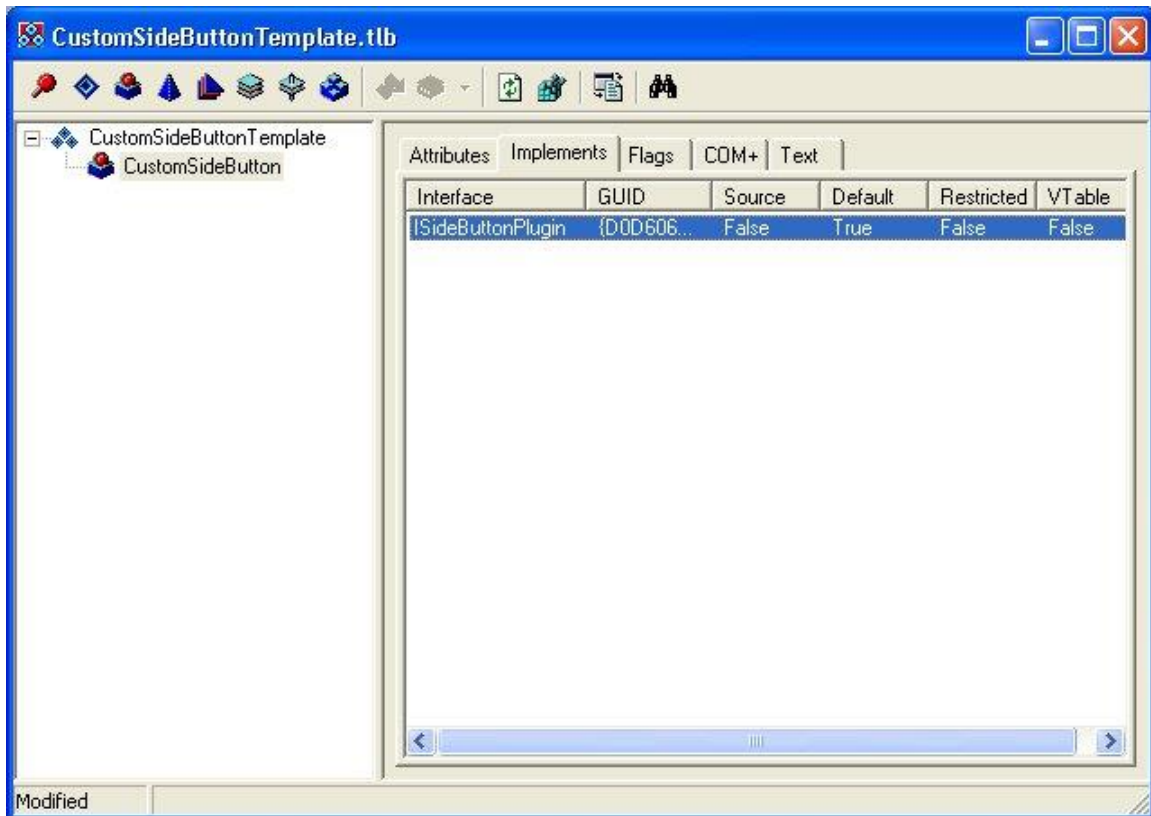
1. Click the CoClass button and enter the name of your class as it is defined in your application, minus the initial "T". In other words, for a Plugin class called TCustomSideButton, you would create a CoClass entry called CustomSideButton. The CoClass is the name your Plugin class is known by in Windows.



2. On the Implements page, right click on the empty grid, click Insert Interface, and specify the interface this CoClass implements. In our example, we'll use *ISideButtonPlugin*.



*Result:* The interface is added to the list for your CoClass.



3. Repeat for each Plugin class in your COM server.
4. Proceed to creating the Plugin unit.

## Creating the Plugin Unit

You need to create a new unit to contain the classes that will serve as Plugins.

**Important!** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

### **To create the Plugin unit:**

1. Select File | New | Unit.
2. In the Interface section "uses" clause, include the following modules:

```
uses
{ VCL }
SysUtils, Variants, Windows, Classes, ComObj, ActiveX, StdVcl,
Dialogs, ExtCtrls, AxCtrls,

{ Local type libraries }
Plugins_TLB,
CustomSideButtonTemplate_TLB;
```

**Note:** If you plan on implementing the IPrintPlugin interface, you'll also need MSXML\_TLB.

3. Proceed to creating the Plugin class.

## Creating the Plugin Classes

**Important!** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

### To create the Plugin class:

1. Within the Plugin unit, define a class that will be used as a Plugin. The definition should look similar to the following. Copy this code snippet and paste it into the new unit after the "type" section keyword:

```
type
  TCustomSideButton = class( TAutoObject,
                           IAbstractPlugin,
                           IBOPlugin,
                           ISideButtonPlugin )

private
  // Data members for properties
  fDescription: String;
  fGUID: TGUID;
  fBusinessObjectType: Integer;
  fAdapter: IPluginAdapter;
  fPriority: PluginPriority;
  fButtonEnabled: WordBool;
  fEnabled: WordBool;
  fHint: string;
  fLayoutName: string;

protected
  // Property getters and setters
  function Get_Description: PChar; safecall;
  function Get_GUID: TGUID; safecall;
  function Get_BusinessObjectType: Integer; safecall;
  function Get_Adapter: IPluginAdapter; safecall;
  function Get_Priority: PluginPriority; safecall;
  function Get_Caption: PChar; safecall;
  function Get_PictureFilename: PChar; safecall;
  function Get_ButtonEnabled: WordBool; safecall;
  function Get_UseLayoutManager: WordBool; safecall;
  function Get_Checked: WordBool; safecall;
  function Get_Hint: PChar; safecall;
  function Get_LayoutActionName: PChar; safecall;
  function Get_Enabled: WordBool; safecall;
  procedure Set_Adapter(const Param1: IPluginAdapter); safecall;
  procedure Set_Enabled(Value: WordBool); safecall;

public
  (** In a side button Plugin, Description is only displayed
    by Retail Pro in debug windows, such as that shown when
    /LOGPLUGLOAD is specified on the command line.
  )

  property Description: PChar read Get_Description;

  (** The GUID is the Globally Unique Identifier used as the
    class ID (CLSID) to identify this class to the Windows
    COM layer.
  )
  property GUID: TGUID read Get_GUID;

  (** BusinessObjectType is the numeric identifier for the
    business object this side button should be associated
    with. When a frame that uses this business object is
    presented, it will have added to its list of buttons,
    in the Menu Designer, an action that references this
    button. Selecting that action causes this button to
    be displayed on the side menu.
  )
  property BusinessObjectType: Integer read Get_BusinessObjectType;

  (** Set by RetailPro, this is a reference to the adapter
    instance within RetailPro that this button can use
    to gain access to the resources, data, and functions
  )
```

```

        within the application.
    }
property Adapter: IPluginAdapter write Set_Adapter;

    /** Priority controls, in a limited fashion, the order
        in which the Plugins are loaded at the time they're
        activated. The three settings for this property are
        high, normal, and low.
    */
}
property Priority: PluginPriority read Get_Priority;

    /** Caption contains the text displayed on the button
        face.
    */
}
property Caption: PChar read Get_Caption;

    /** PictureFilename is the name of a BMP file on disk that
        is used as the glyph for the button face.
    */
}
property PictureFilename: PChar read Get_PictureFilename;

    /** ButtonEnabled controls whether the button is available
        or grayed out. This property is checked several times
        a second, allowing the Plugin to react to changes in
        the application, so the logic within the Get_ButtonEnabled
        getter must be succinct.
    */
}
property ButtonEnabled: WordBool read Get_ButtonEnabled;

    /** This property controls whether the button can be added
        to a side menu via the layout manager. This should
        always be set to TRUE.
    */
}
property UseLayoutManager: WordBool read Get_UseLayoutManager;

    /** This controls whether the button face includes a check
        mark.
    */
}
property Checked: WordBool read Get_Checked;

    /** Hint contains the fly-over help text shown to the user
        when the mouse idles over the button control.
    */
}
property Hint: PChar read Get_Hint;

    /** LayoutActionName is the text used to name the action
        instance.
    */
}
property LayoutActionName: PChar read Get_LayoutActionName;

    /** This controls whether the Plugin is enabled, not whether
        the button shows as enabled. Typically, this is
        set to True, and only turned off, by RetailPro, if the
        button's HandleEvent event handler returns an exception
        to the application.
    */
}
property Enabled: WordBool read Get_Enabled write Set_Enabled;

    /** Prepare is called by RetailPro to signal to the Plugin
        class that it is about to be enabled and needs to perform
        whatever logic is required to initialize it.
    */
}
function Prepare : WordBool; safecall;

    /** This method is used to set the data members used to
        identify the Plugin, but no class or memory allocation
        should be done here. Initialize is called by the COM
        layer anytime the Plugin is instantiated. Prepare
        is called before a Plugin is actually activated
        for use by the UI, and that's where connections can be

```

```
        opened, memory allocated, etc.
    }
    procedure Initialize; override;

    (** HandleEvent is the equivalent of the OnClick event handler
        on a TButton control.
    )
    function HandleEvent : WORDBOOL; safecall;

    (** CleanUp is called to reverse the work done in the Prepare
        method. It is called only when a Plugin has been
        activated for use by the UI and is about to be discarded.
        NOTE: The reference to the Adapter must be set to NIL in
        this method, along with any other pointers to allocated
        or external objects and memory blocks. Failure to do so
        can result in a hung application and critical exceptions
        being raised.
    )
    procedure CleanUp; safecall;
end;
```

Before proceeding, place your cursor inside the class definition and press *Ctrl+Shift+C* to use Delphi's class completion feature. The class implementation will be created for you.

### Important Requirements

- The Plugin class must be a descendant of TAutoObject, which is Borland's quick and easy class for defining automation objects that will be instantiated by a COM server.
- The Plugin class must also specify the interface that you wish to implement and the interfaces that it inherits from. This probably seems redundant, but the "Supports" routine can only detect interfaces that are specifically declared in the class definition, and those interfaces are used by Delphi components and routines to validate access to an interface reference. If you plan on creating a validation routine, this would necessarily mean that you would need to include the interfaces shown in the example above.
- The "private" section of the example shows the data members used to store the values of the properties defined as part of the implementation of the various interfaces listed.
- The "protected" section contains the *getters* and *setters* for the properties. Note that these are the routines defined in the interfaces. Delphi doesn't care that they are defined as protected. These HAVE to be redeclared in the user's class and implemented.
- The "public" section defines this class as it appears to the routines that make use of it. The following shows the methods defined in the public section as they relate to our example.



**Method: procedure Initialize; override;**

Code	Notes
<pre> procedure TCustomSideButton.Initialize; begin     inherited;     fDescription := 'My Custom Side Button';     fGUID := CLASS_CustomSideButton;     fBusinessObjectType := btInventoryItems ;     fPriority := ppNormal;     fEnabled := true;     fButtonEnabled := true;     fHint := Click Me';     fLayoutName := 'CustomSideButton'; end; </pre>	<p>Overridden from TAutoObject's Initialize method, this can be used to set the data member's initial values.</p> <p>The CLASS_CustomSideButton constant is generated when Delphi creates the .PAS module for the Plugin's local type library. btInventoryItems is defined in the Plugins Type Library as part of the enumerated type BusinessObjectType. The attribute name is from Retail Pro®'s repository definition of the Inventory Items business object.</p>

**function HandleEvent : WordBool; safecall;**

This function is called when the Plugin is expected to handle some event, such as validating a field value.

**Code**

```
Function TMyValidationPlugin.HandleEvent : WordBool;
Const
    sMZWarning = 'WARNING: Customers with last names in the range M-Z'
                + 'are to be referred to department B';
Var
    varValue : Variant;
    stValue  : String;
begin
    result := False;

    // For classes that implement IBOPlugin, handle zero
    // is always the primary business object. In our case,
    // it's the Customer business object because we specified
    // btCustomer as the BusinessObjectType property's value.

    varValue := fAdapter.BOGetAttributeValueByName(0, 'Last Name');

    // In Delphi, conversion from an OLEVariant to a Dephi
    // Variant occurs
    // automatically. Check to make sure the value is not empty or
    // null.
    if ((not VarIsEmpty(varValue))
    and (not VarIsNull(varValue))) then
    begin
        // Take the value, check it against some criteria,
        // set values and/or display a message if necessary.
        stValue := varValue;
        stValue := trim(uppercase(stValue));
        if (copy(stValue, 1, 1) > 'M') then
        begin
            varValue := True;
            varValue := fAdapter.BOGetAttributeValueByName(0, 'Last Name');
            result := True;
            ShowMessage(sMZWarning);
        end;
    end;
end;
```

In our example, we'll use a simple one:

```
function TCustomSideButton.HandleEvent : WordBool;
begin
    result := False;
    showmessage( 'Button Clicked' );
end;
```

**Method: procedure Cleanup; safecall**

This method is called when the Plugin is expected to prepare for being destroyed.

**Important!:** You must set the Adapter to NIL. The COM facility will not signal that the reference to the application has been cleared if you do not perform this operation. Consequently, even though the adapter is freed, it is not cleared from memory.

**Code**

```
procedure TCustomSideButton.Cleanup;
begin
    fAdapter := NIL; // it's critical that you NIL this reference before leaving!
end;
```

**Method: function Prepare: WordBool; safecall;**

This method is called when the Plugin has been instantiated and is used in regular processing (as opposed to discovery).

**Code**

```
function TCustomSideButton.Prepare: WordBool;
begin
    result := TRUE;
end;
```

**Method: property Description: PChar read Get\_Description;**

This is a short text description, usually just an identifying name that is used for debugging purposes and for display when editing preferences.

**Code**

```
function TCustomSideButton.Get_Description: PChar;
begin
    result := PChar( fDescription );
end;
```

**Method: property GUID: TGUID read Get\_GUID;**

This is a system-unique GUID. In Delphi, this value is generated automatically when you create a type library and assigned a name like CLASS\_MyValidationPlugin. If you are using Delphi, you should use the constant name assigned by Delphi.

**Code**

```
function TCustomSideButton.Get_GUID: TGUID;
begin
    result := fGUID;
end;
```

**Method: property BusinessObjectType: Integer read Get\_BusinessObjectType;**

This is one of any of the constants related to the enumerated type BusinessObjectType in the Plugins type library.

**Code**

```
function TCustomSideButton.Get_BusinessObjectType: Integer;
begin
    result := fBusinessObjectType;
end;
```

**Method: property AttributeName: PChar read Get\_AttributeName write Set\_AttributeName;**

This is the name of the field for which the validation Plugin should be triggered when changes are made.

**Code**

```
function TCustomSideButton.Get_Attribute : String;
begin
    result := pchar( fAttributeName );
end;
```

**Method: property Adapter: IPluginAdapter write Set\_Adapter;**

This property is set by the Plugin manager prior to the call to Prepare.

**Code**

```
procedure TCustomSideButton.Set_Adapter(const Param1: IPluginAdapter);
begin
    fAdapter := Param1;
end;
```

### Method: **property Priority: PluginPriority read Get\_Priority;**

This property allows a Plugin limited control over the order in which the Plugins are instantiated and prepared. The three settings are ppHigh, ppNormal, and ppLow. Within each level, instantiation occurs by COM server in the order in which the Plugin discovery occurred – first come, first served.

#### Code

```
function TCustomSideButton.Get_Priority: PluginPriority;
begin
    result := fPriority;
end;
```

## Creating the Discovery Class

For the Plugin manager to obtain information about the Plugins in the COM server, there must be at least one class defined in the COM server that implements the **IDiscover** interface. This interface is used solely during Retail Pro startup and provides Retail Pro with all the information needed to load the Plugins.

### function **PluginGUIDs: OleVariant**

This method returns a text (string) GUID for the plug-in.

### function **PluginModuleVersion: PChar**

This method returns the revision of the plug-in.

### procedure **APIVersion(out MajorVersion: Integer; out MinorVersion: Integer; out Revision: Integer)**

This methods returns in three parts the revision of the API that the plug-in is using. Unless otherwise noted this should be as follows.

```
MajorVersion = 1
MinorVersion = 0
Revision = 0
```

**Important!:** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

1. Create a single class for the COM Server object that implements the IDiscover interface. The IDiscover interface looks like this (in Delphi):

```
TCustomSideButtonDiscover = class( TAutoObject, IDiscover )
public
    (** APIVersion will, in future versions of the PI-API, be used
        to determine if a Plugin can be loaded and, if it can but
        uses an older version of the API, how to handle backwards
        compatibility where possible.
    **)
    procedure APIVersion(out MajorVersion: Integer; out MinorVersion: Integer; out
Revision: Integer); safecall;

    (** This is a list of the GUIDs for every class, besides the
        discovery class itself that will be used by Retail Pro.
        These classes are each instantiated and queried as part of
        the discovery process.
    **)
end;
```

```
    }  
function PluginGUIDs: OleVariant; safecall;  
  
    (** This is a purely informational bit of text, used to identify  
        a Plugin module.  
    )  
function PluginModuleVersion: PChar; safecall;  
end;
```

### **How the Discovery Class is Used**

How the Discovery class is used depends on whether you are using a manifest file or registering the COM class using regsvr32.

When the Plugin manager has read the server and discovery class information from the manifest file, it uses the server and discovery class name to request the class ID from Windows for that class.

When regsvr32 is used to register a COM class, information about that COM class is saved in the registry under HKEY\_CLASSES\_ROOT. The retrieved class ID is then used to instantiate the discovery class, using the following procedure:

1. The supported API version is requested. This is checked against the application's API version to see if the Plugin can be supported. If not, the COM server's discovery object is freed and no further action is taken for Plugins in that COM server module.
2. The application requests a variant array that contains string representations of the GUIDs for each Plugin class in the COM server.
3. The Plugin manager runs through that list, instantiates each Plugin, and determines the type of the Plugin by querying it to see what interface it supports.

Each Plugin is queried for contextual information, specifically which business object it needs to be attached to, and depending on the type of Plugin, the form name, attribute name, caption, or hardware type. This information is stored for later use, and the Plugin is then freed.

4. After the GUID array has been processed, the discovery object is freed and the Plugin manager goes on to the next line in the manifest file.

### **Discovery Class Methods**

The Discovery class includes the following methods:

**procedure APIVersion (out MajorVersion: Integer; out MinorVersion: Integer; out Revision: Integer); safecall;**

This routine hands back the version (in three parts) that identifies which version of the Plugin API this Plugin implements and expects. (As of this writing, the routine should hand back 1, 0, and 0 as the values of the output parameters, respectively). If the version handed back to the Plugin manager is not supported by Retail Pro, the COM server and its Plugins are ignored. No error is reported. For best practices; the constants PluginsMajorVersion and PluginsMinorVersion (see Plugins\_TLB.pas file) should be used to pass back. This will ensure that your Plugin will report back to RetailPro which Plugins.Tlb file it was compiled against.

```
procedure TCustomSideButtonDiscover.APIVersion( out MajorVersion,
                                                MinorVersion,
                                                Revision: Integer);
begin
    MajorVersion := PluginsMajorVersion;
    MinorVersion := PluginsMajorVersion;
    Revision     := 0;
end;
```

### **function PluginGUIDs: OleVariant; safecall;**

This routine must hand back an OleVariant list of OleVariants, each containing a character string representation of the GUID for one of the Plugins in this module, minus the discovery Plugin.

```
function TCustomSideButtonDiscover.PluginGUIDs: OleVariant;
var
    tmp : OleVariant;
begin
    // Here's where we build an array of GUIDs to pass back to the
    // PluginManager. If you add an entry to this array, be sure to
    // go to the initialization section of this unit and add a line to
    // register the Plugin class with the Windows COM service.
    // Create an array with bounds from 1 to 1 (because we're only
    // defining one Plugin for this example).
    // If you add a Plugin, increment the high bound
    // in this create statement.
    result := VarArrayCreate( [ 1, 1 ], varOleStr );

    // Convert the GUID to a character string. Delphi converts this
    // automatically to an OLE variation.
    tmp := GUIDtoString( CLASS_CustomSideButton );
    result[ 1 ] := tmp;

    // The operation above would be repeated for each Plugin.
end;
```

### **function TMyDiscover.PluginModuleVersion: PChar;**

This returns a string that can be used for reporting back to the user during configuration or can be used for debugging purposes. This value does not affect Retail Pro processing in any way.

```
function TCustomSideButtonDiscover.PluginModuleVersion: PChar;
begin
    result := '3.2.1 (beta)';
end;
```

## Creating Instances of the Class Factory for each Plugin

**Important!** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

Delphi's ComServ unit defines the class factory type needed to create and deliver the Plugin classes to Retail Pro®, and the next step is to create a class factory for each Plugin class. In Delphi, this would be in the initialization section of one of the units in the COM server or in the DPR's `begin..end` block.

If you created a separate unit for the discovery class, add the class factory to the discovery units Initialization section.

### To create a class factory for each Plugin:

1. Insert code similar to that below in a place where it can be executed when the COM server DLL is started up. In Delphi, this would be in the initialization section of one of the units in the COM server or in the DPR's `begin..end` block. For our example, we'll place this in the `CustomSideButtonDiscover.pas` file.

```
try
    TAutoObjectFactory.Create(
        ComServer,
        TCustomSideButtonDiscover,
        CLASS_CustomSideButtonDiscover,
        ciMultiInstance,
        tmApartment);
except
    on E1:Exception do
        raise Exception.Create( E1.Classname + ' exception registering the
Custom Side Button Discover class: ' + E1.Message );
end; //try

try
    TAutoObjectFactory.Create(
        ComServer,
        TCustomSideButton,
        CLASS_CustomSideButton,
        ciMultiInstance,
        tmApartment);
except
    on E2:Exception do
        raise Exception.Create( E2.Classname + ' exception registering the
Custom Side Button class: ' + E2.Message );
end; //try
```

**Note:** A factory is created for the discovery class as well. Without these calls, COM will not see class factories in the COM server module and will not be able to instantiate them either to register them with the system or to deliver them to Retail Pro®.

2. Proceed to exporting procedures.



## Exporting Procedures

After you have created instances of the class factory for each Plugin, you need to export certain procedures.

**Important!:** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

1. Export the following procedures:

- DllGetClassObject
- DllCanUnloadNow
- DllRegisterServer
- DllUnregisterServer

No other exports are required or supported.

*Note:* If you followed the basic steps for creating a Plugin using Delphi, these exports are already defined in the DPR module.

2. Compile your project and copy the resulting DLL to the *RetailPro9\Plugins* directory.
3. Proceed to registering the COM servers.

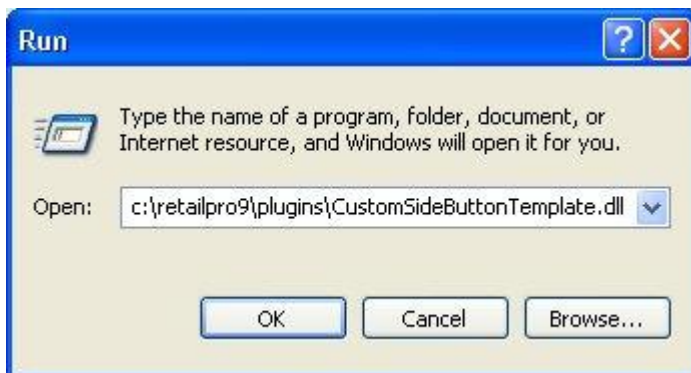
## Registering the COM Servers

**Important!:** These instructions assume that you are using Delphi to create the COM server and Plugins. If you decide to create a COM server using a different development environment, please contact the development staff at RetailPro, Inc.

After you've put the finishing touches on the Plugin and compiled it, you will need to register it with Windows. Place it in the \RetailPro9\Plugins\ directory.

### **To register the COM server:**

1. Select Start > Run. The Run dialog box is displayed. Type regsvr32 [<path>\dllfilename], and then click OK <Enter>. The Plugin dll is registered with Windows.



To unregister a server:

1. Select Start > Run.
2. Type regsvr32 [<path>\dllfilename] -u, and then click OK **<Enter>**.

### Notes about Regsvr32

Regsvr32 is a Windows utility. Regsvr32 loads the DLL and calls the exported method DllRegister (or DllUnregister if the "-u" flag was specified). Everything else that occurs is controlled by the DllRegister procedure in the DLL, which is defined in Delphi's ComServ unit. When the DLL is loaded, the initialization sections for each unit are executed, including the logic that creates the TAutoObjectFactory objects for each CoClass. Once the DLL is loaded, the DllRegister method is called, and this writes the necessary information into the registry for each class that a TAutoObjectFactory object was created for.

**Note:** If the new Plugin is installed on a machine on which the Plugins.tlb file has not been registered, it will not register and will not function correctly. In this case, Borland's regsvr.exe utility can be used to register the Plugins type library first.

## Manifest Files

For the Plugin manager to see the COM server, there must be a manifest (.mnf) file in the \RetailPro9\Plugins directory. This manifest lists the names of the classes in the COM server module that implement the IDiscover interface. The name must appear in the manifest exactly as it does in the registry following the use of regsvr32 to register the Plugin. If you're not sure what to put in the manifest, then register the Plugin and then do a search in the registry using regedit for the name of your Plugin.

**Warning!** Using the regedit program should be done with the utmost caution. You can do serious damage to your system if you are not careful. Each COM server must be represented by a single line in a manifest, and must be formatted as follows:

```
Servername.DiscoverClassName[:CLSID={000...}][,AutoReg=Y]
```

Example:

```
MyServer.MyDiscoverPlugin:CLSID={3C97FFAF-22AD-4CC5-9FC8-885874F15805}, AutoReg=Y
```

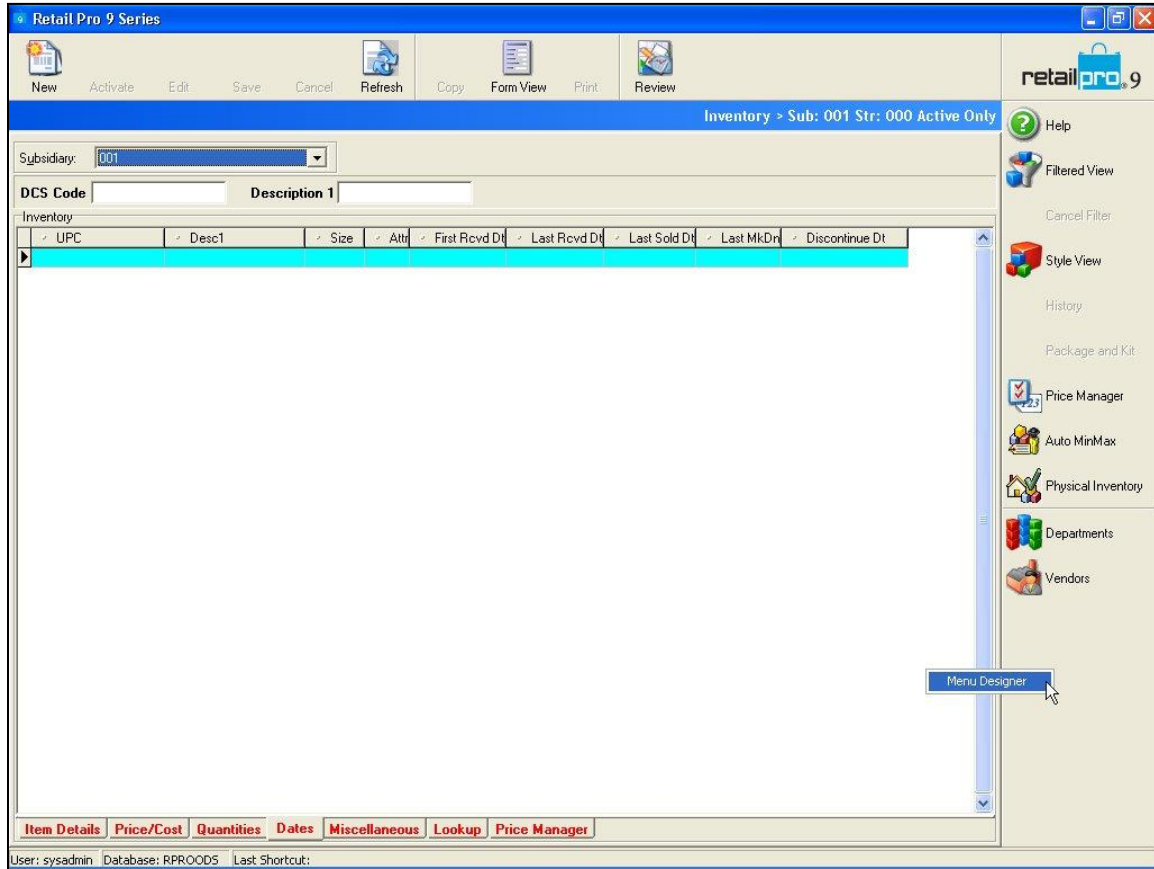
Servername	The name of the server module. Typically, this will be the name of the server without the dll extension
DiscoverClassName	The name of the class in the COM server module that implements the IDiscover interface. This is the name used in the Plugin's type library.
CLSID	Class ID for the Plugin that implements the IDiscover interface, defined in the Plugin project's type library. (optional)
AutoReg	Indicates whether or not to auto-register the server if it's not already registered with Windows. Note that this can only be specified if the CLSID qualifier is present. If this qualifier is enabled, RetailPro will first check to see if the Plugin is registered and, if it's not, RetailPro will register the server, then proceed with processing the discovery Plugin for that server.

## Test Your Plugin

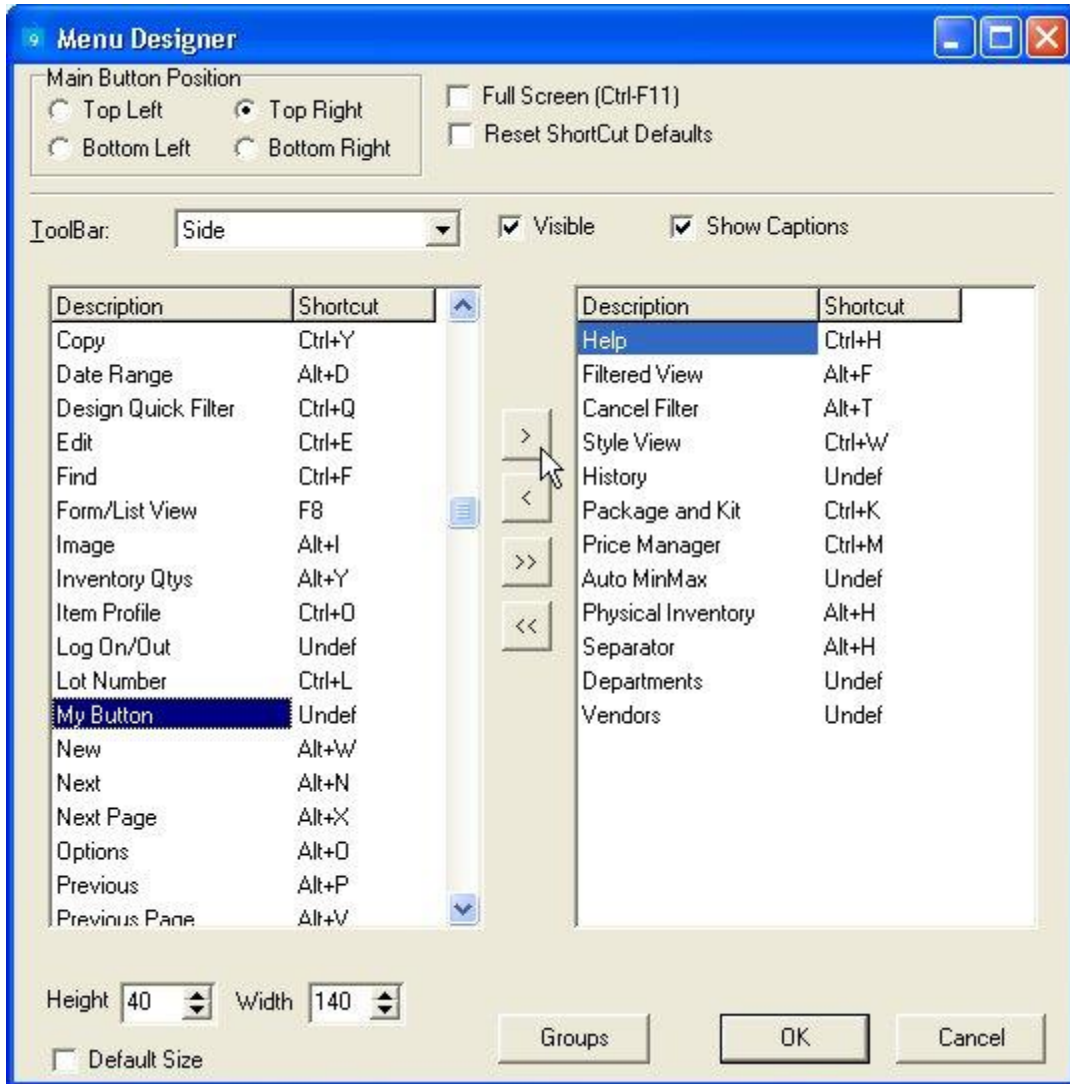
Now we need to find out if our Plugin works. Be sure your Plugin and the matching manifest files are in the RetailPro9\Plugins directory before you begin.

1. Open RetailPro. The splash screen advises that Plugins are being loaded. If unsuccessful, a message dialog is displayed and you'll need to double check that your manifest file is correctly configured.
2. When the main screen of RetailPro appears, ensure you have a menu button at the top for Inventory. If not:
  - a. Right-click in the top menu area and select Menu Designer.
  - b. In the left pane of the Menu Designer, select Inventory.
  - c. Click the '>' button to move it to the right pane.
  - d. Click OK.
3. Click the Inventory menu button to display the Inventory screen.

4. Right-click in the side menu and then click on the Menu Designer.

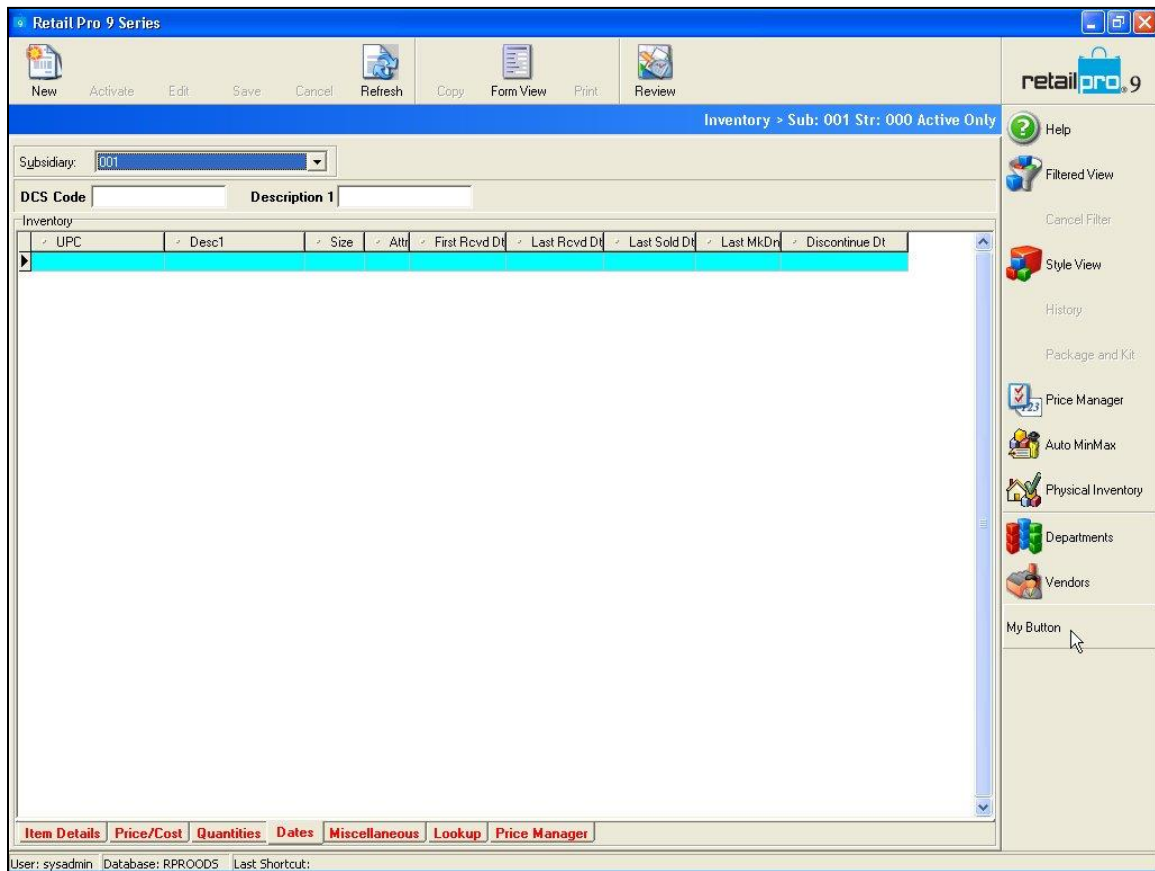


The menu designer dialog is displayed.



5. Find your new button in the left pane list.
6. Select your button and then click the '>' button.
7. Click OK.

8. Your button now appears in the side menu of RetailPro. Try it to see if it works.



## Error Handling

In all situations, Retail Pro 9 is shielded as much as possible from errors caused by a given Plugin. The base adapter class in Retail Pro includes a generic method for recording and discarding exceptions generated by the Plugin. Plugins as a rule are not allowed to disrupt the processing of Retail Pro 9 for any reason. To this end, all exceptions generated by the Plugins, if they do make it back to RetailPro, are captured, logged if necessary, and handled in such a way that the Plugin is either reset or disabled.

Where possible, all values going to and from the Plugins are validated for the following conditions:

- Value range
- Buffer size
- NIL pointers where NIL is not a valid parameter

As a best-practice, Plugins should attempt to do the same. You are encouraged to implement thorough, configurable logging capabilities, and you should program defensively, testing all parameters before working with them and capturing any exception that might occur. Since some exceptions crossing the COM barrier between the Plugin and the Retail Pro application may cause the entire application to shut down and be removed from memory by the Windows operating system, you should attempt to avoid all exception causing scenarios and code in logic to trap all exceptions, log them, and pass back the appropriate error code to Retail Pro to express the error. See Error Codes for a list of error codes included in the Plugins type library (Plugins.tlb).

## Tips and Tricks

### Architecture Tips

It is recommended that you divide the modules that make up your Plugin the following fashion:

1. Have one module that acts as the trap for calls from the application.
2. Put screens that are displayed in separate modules.
3. The discovery class should be defined in its own module.
4. Include an "Enabled" flag for each Plugin in the Plugin's configuration.
5. If a Plugin is not enabled, it should use no memory and should return all calls as unprocessed.

### Exceptions

Plugins should NEVER let an exception surface beyond the Plugin. Handle the exception and return an error code of some kind or False. Additionally, you should pop up a window informing the user what happened and how to fix it, and then call DisablePlugin.

You should pay particular attention to Variants. Casting a variant incorrectly will cause the PluginManager to halt processing of your Plugin's requests. It is recommend that at a minimum you use a Try...Except on "EVariantTypeCastError" within the "HandleEvent" section of your Plugin.

## Logs

If your Plugin needs to generate logs, do not place the logs in the Plugins directory. Logs should be stored in the \RetailPro9\Logs directory. You should open the log with a filename format similar to:

PluginIdentifier\_Workstation\_yyyymmdd.log

Where:

PluginIdentifier = Any string that uniquely identifies your Plugin or Plugin set.

Workstation = a unique identifier for the machine which the Plugin-in running on. If this is not Specified in the filename, you should include it as part of the log detail.

Yymmdd = Date log is opened. Times should be noted as part of the log detail.

## Sample Code

The code below should help you to understand how to create, locate and retrieve BO data. The trick about BOs is that, while we treat them like datasets, they aren't. They have predefined SQL statements that sometimes include filters that must be satisfied. Usually the only filter we need to worry about is the subsidiary number. If the "Sbs No" is left null, then, whether you like it or not, you're filtering on Sbs\_No = null (see "BO Filtering requirements" below).

### **To determine the Subsidiary Number of the BO of type "btInvoice":**

```
SbsNo := FAdapter.BOGetAttributeValueByName( 0 , PChar('Subsidiary
Number') );
CustID := FAdapter.BOGetAttributeValueByName( 0, 'STC Cust #' );
```

### **To create a new BO ( btCustomer ):**

```
CustRef := FAdapter.CreateBOByID( btCustomer );
```

Make sure to filter the new BO on the current Subsidiary Number before "BOOpen"

NOTE: 'Sbs No' is the name of the "Customer" subsidiary attribute name, not 'Subsidiary Number' which is the "Invoice" subsidiary attribute name.

```
FAdapter.BOSetFilterAttr( CustRef, pchar( 'Sbs No' ), SbsNo, proEqual,
True, True );
```

```
FAdapter.BOOpen( CustRef );
```

To Retrieve BO ( btCustomer ) data:

```
AttributeNames := VarArrayCreate( [ 1, 1 ], varOleStr );
AttributeValues := VarArrayCreate( [ 1, 1 ], varOleStr );
```

```
AttributeNames[1] := 'Cust ID';
AttributeValues[1] := CustID;
```

```
ErrorId := FAdapter.BOLocateByAttributes( CustRef, AttributeNames,
AttributeValues);
```

```
if ErrorId = peSuccess then begin
```

```
    ZipCode := FAdapter.BOGetAttributeValueByName( 0, 'ZIP' );
```

```
.....
```



### Dealing with var PChar's Parameters:

Some PIAPI Methods have ( **var** AParameter: **PChar** ) parameters, which can be tricky. As in the Delphi documentation says in "Returning a PChar local variable", don't do this:

```
Procedure MyMethod( var AParam: PChar ) ;
Var
    ALocalStr : string;
Begin
    ALocalStr := 'Have a nice Day';
    AParam := PChar( ALocalStr );
End;
```

What you are essentially doing is assigning the AParam pointer to the ALocalStr address which is promptly destroyed when MyMethod loses scope. For example, the **ItemAddRemovePlugin.BeforeChooseProcessItem** method has a parameter: **var AItemData: PChar**. Below is the recommend method for dealing with this parameter type.

```
function TItemAddRemovePlugin.BeforeChooseProcessItem(
    ItemBOHandle : Integer;
    ADocSID      : PChar;
    var AItemData : PChar;
    var AModified : WordBool
) : WordBool;
begin
    result := TRUE;
    if ( fEnabled ) then
        begin
            ...
            ...do some work.
            ...
            FItemData := AItemData;
            ...
            ...do some work.
            ...
            If TheWorkModifiedFItemData then
                begin
                    //Make sure to set AModified to True if indeed you are Modifying AItemData
                    AModified := True;
                    // This will copy FItemData into the AItemData location.
                    StrLCopy( AItemData, PChar( fItemData ), Length( AItemData ));
                end;
        end;
    End;
End;
```

## Troubleshooting

<i>Problem</i>	<i>Solution</i>
The application exits but stays in memory	<p>Set all references to any object or structure in the application to NIL. Failure to do this will cause COM to hold the object in memory indefinitely. That means anything connected with it including the application itself. Typical references:</p> <ul style="list-style-type: none"> <li>• The Adapter property</li> <li>• References to other Plugins</li> <li>• PChars</li> </ul>
The application starts up, then goes away	<p>This is typically caused when a type library (such as Plugins.tlb) that is required by the Plugin hasn't been registered on that machine the Plugin is running on. Use the tregsvr.exe that was installed in the RetailPro directory to register the type library – Windows' "regsvr32" cannot be used to register the type library.</p>
I keep getting peInvalidBOHandle when I try to access the BOxxx	<p>If the Plugin you wrote implements an interface that descended from IBOPugin, zero will always be the handle for business object that you specified in your Business Object property. Additional business objects can be opened by your Plugin, and handles will be passed back for those business objects. You can also obtain handles for parent business objects (GetHandleForRootBO) and reference business objects (GetReferenceBOForAttribute). Sometimes you may want to create a business object in Plugin #1, then share that created business object with Plugin #2. If Plugin #2 doesn't clone the handle received from Plugin #1, it won't be able to see the business object. See CloneBOHandle for details.</p>
The hardware Plugin appears to be discovered ok, but when I click "Configure", the exception says the class isn't registered.	<p>Quite often, when copying from an existing project, we've found that folks tend to forget that the GUID property on the Plugin object must return the correct GUID for that class. If the discovery class specifies a given Class ID for an object, the GUID property on that object must return the same Class ID.</p>
I am trying to get to the invoice items from an invoice BO, but when I use CreateBOByID it returns -3 (not a valid BO handle).	<p>Some BO types, like the invoice items BO, can't be created by themselves because they are designed to be detail BOs in a master-detail arrangement, therefore they require a parent BO. You can use CanBeCreated to determine whether a given BO type can be created using CreateBOByID or CreateBOByName.</p>

## Hardware Plugins

Hardware control in Retail Pro 9 is similar to Retail Pro 8. While the methods, properties, and events described here use the UPOS specification in combination with the 8 series architecture, the Retail Pro 9 architecture makes no assumptions that POS drivers are being used (except with printers)

Logic is not built directly into Retail Pro 9 to handle special case drivers and direct device access – all such functionality should be addressed by Plugins that are custom built. The Plugins must translate a general method call from the application into the necessary series of calls (for example, OPOS calls) and to return the results in the expected format.

The following table describes the devices that Retail Pro 9 supports and the type of support. “Wedge” is a term that means that the device plugs in between the keyboard and the computer – output appears to the application (and to Windows) as though it comes from the keyboard, so additional support is required.

All hardware support is built into Plugins, and as such, the core Retail Pro application is unconcerned with the hardware interface (except for POS printers). All hardware Plugins provided with Retail Pro use OPOS to communicate with the point of sale peripherals.

<i>Device</i>	<i>OPOS via Plugin</i>	<i>Wedge</i>	<i>Windows Driver</i>	<i>Serial Port via Plugin</i>	<i>LPT via Plugin</i>
Line Display	X			X	
Cash Drawer	X			X	X
MSR	X	X			
MICR	X	X			
PIN Pad	X				
Scale	X				
Bar Code Scan	X	X			
Signature Scan	X				
Check Image Scanner	X				
Inventory Scan				X	
Fiscal Printer				X	

\*Support also exists for Ingenico's proprietary forms-based signature capture devices

## Tender and EFT Plugins

You have (at least) three options when it comes to implementing plugins:

- Create IEFTPlugin interface
- Create ITenderDialogue interface
- Create ISideButton interface

### *IEFTPlugin interface*

The IEFTPlugin interface is the way that Retail Pro Core implements ALL of EFT tender plugins. The IEFTPlugin interface is a little more complicated to implement than using ITenderDialogue, but I do not believe that it provides any additional functionality. The IEFTPlugin interface option can only implement the four "EFT" tenders (Credit, Debit, Gift Card, Check). There is no way to "share" a tender button with another (or existing) implementation.

### *ITenderDialogue interface*

The ITenderDialogue interface is probably the most common way that non-Core (custom) tenders are implemented. There are a couple of quirks in the interface but it works fine. This option allows you to override ANY tender type, not just EFT tender types. This option MUST override (replace) one of the existing tenders. There is no way to "share" a tender button with another (or existing) implementation.

### *ISideButton interface*

The ISideButton interface is the easiest to implement but it does not provide a tender button in the same way (which some end users do not like). This is the ONLY option that allows all existing tender types to still be used. We recommend you avoid using the ISideButton interface – unless you need to preserve all existing tenders – because the tender (custom side) button is not part of the existing tender grid.

A merchant usually only uses one gift card provider (or a provider that supports multiple card types). As a result, it usually isn't important to preserve all tenders (except possibly during a changeover period from one provider to another).

### **IEntityUpdate Plugin for Gift Card Activation, Add Value**

You will also want to implement at least an IEntityUpdate plugin on the invoice and to do the actual activation of new gift cards or adding value of existing cards in BeforeUpdate - not in the tender processing.

You may want to implement a custom side button for card balance inquires and possibly balance transfers between cards.

### **IEntityUpdatePlugin to Validate Line Items**

You may also want to implement an IEntityUpdate or IItemAddRemove at the item level to validate cards being sold as line items. Validate that they are in a valid state (not already used, etc.) when they are scanned. Selling them as line items is the best way to sell a bunch of them. Make sure to wait to do the activation until Invoice BeforeUpdate - after you have received actual payment for them.

Selling multiple gift cards should be done by adding them as line items.

You can sell a gift card via the gift card tender processing but that should only be used for one or two cards on a transaction as you cannot track and report on the card numbers easily and there is limited storage space in the tender.

Part of that is specific to implementing GIFT CARD processing and part is specific to implementing any non-core tender.

# Retail Pro 9 Hardware Interface Implementations

The following interface implementations have been added to the Plugins type library (**Plugins.tlb**).

*Note:* For the most up-to-date details on the interfaces, please refer to the **Plugins.tlb**.

## IDisplayPlugin

### IDisplayPlugin.ClearDisplay

```
function ClearDisplay : Integer;
```

Tells the Plugin to clear the entire line display read-out. To clear a single line, see **ClearText**.

### IDisplayPlugin.ClearText

```
function ClearText(Line : Integer) : Integer;
```

Tells the Plugin to clear only the line specified. To clear the entire display, see **ClearDisplay**.

### IDisplayPlugin.DisplayText

```
function DisplayText(Line : Integer; const AText : String; TextPlace : TLineDisplayTextPlacement) : Integer;
```

This tells the Plugin to display a single line of text and specifies whether to right justify, left justify, or center the text.

## ICashDrawerPlugin

### ICashDrawerPlugin.IsDrawerOpened

```
function IsDrawerOpened: Boolean;
```

When called, the Plugin must indicate whether the drawer is currently open. If the drawer state cannot be determined, it is recommended that you always return **False**.

### ICashDrawerPlugin.OpenDrawer

```
function OpenDrawer : Integer;
```

Signals the Plugin to tell the cash drawer to open.

### ICashDrawerPlugin.DrawerNumber

```
property DrawerNumber : Integer;
```

This is set by RetailPro to the number of the drawer on which the Plugin should operate.

### **ICashDrawerPlugin.OnCashDrawerStatusUpdateEvent**

`property OnCashDrawerStatusUpdateEvent;`

Set by RetailPro to an event handler address. When there is a change in status (open/closed), the Plugin should call this event handler.

## **IMSRPlugin**

### **IMSRPlugin.OnMSRDataEvent**

`Property OnMSRDataEvent;`

Set by RetailPro to an event handler address. When the MSR device generates a data event, the Plugin should pass that signal on to this event handler.

## **ICheckImageScannerPlugin**

### **ICheckImageScannerPlugin.AcquireImage**

`Function AcquireImage : Integer;`

This method tells the Plugin to interact with the scanning device to obtain an image of the inserted check. Upon return, the Document property should contain a valid pointer to the image in memory.

### **ICheckImageScannerPlugin.Quality**

`property Quality : Integer; {rw} // dots per inch`

This indicates the dots per inch in an image. Typically set to 100, 200, 300.

### **ICheckImageScannerPlugin.Contrast**

`property Contrast : Integer; {rw}`

The contrast property. Range is between 0 (lightest possible setting) and 100 (darkest possible setting), with 50 being the default setting.

### **ICheckImageScannerPlugin.ImageFormat**

`property ImageFormat : Integer; {rw} // one of enumerated image format constants`

Sets image format to one of the OPOS constants for Native, TIFF, BMP, JPEG, or GIF.

### **ICheckImageScannerPlugin.DocumentHeight**

`property DocumentHeight : Integer; {r} // pixels`

Height of image to scan.

### **ICheckImageScannerPlugin.DocumentWidth**

`property DocumentWidth : Integer; {r} // pixels`

Width of image to scan.

### **ICheckImageScannerPlugin.Status**

```
property Status : Integer; {r} // one of enumerated status constants (empty, ok, full)
```

One of the OPOS constants for Empty (no image in memory), Ok (image ready), or Full (out of memory in device).

### **ICheckImageScannerPlugin.Document**

```
property Document : Pointer; {r} // binary, format depends on ImageFormat property
```

Following a scan, this is the pointer to the image in memory.

## **IMICRPlugin**

### **IMICRPlugin.InsertDocument**

```
function InsertDocument : Integer;
```

This is called by RetailPro to signal that device should be prepared for a document to be inserted and then, once the document has been inserted, to take the device out of insertion mode.

### **IMICRPlugin.RemoveDocument**

```
function RemoveDocument : Integer;
```

This is called by RetailPro to signal that the device should now be prepared to have a previously inserted document removed. Upon removal, it should take the device out of document removal mode.

### **IMICRPlugin.OnMICRDataEvent**

```
property OnMICRDataEvent;
```

Set by RetailPro to an event handler address. When the MICR device signals a data event, the Plugin should pass that signal on to RetailPro.

### **IMICRPlugin.OnMICRStatusUpdateEvent**

```
property OnMICRStatusUpdateEvent;
```

Set by RetailPro to an event handler address. When the MICR device signals a change in status, the Plugin should pass that signal on to RetailPro.

### **IMICRPlugin.AccountNumber**

```
property AccountNumber : String; {r}
```

Set by RetailPro to an event handler address. When the MICR device signals a data event, the Plugin should pass that signal on to RetailPro.

### **IMICRPlugin.Amount**

```
property Amount : Double; {r}
```

If there is an amount included in the MICR data, this property is set by the Plugin to the amount value.



### **IMICRPlugin.BankNumber**

```
property BankNumber : String; {r}
```

The number of the issuing bank, read in by the Plugin from the MICR data.

### **IMICRPlugin.CheckType**

```
property CheckType : Integer; {r} // one of enumerated check types (personal,
business, unknown)
```

The check type parsed from the MICR data, one of the OPOS constants MICR\_CT\_PERSONAL, MICR\_CT\_BUSINESS, or MICR\_CT\_UNKNOWN.

### **IMICRPlugin.CountryCode**

```
property CountryCode : Integer; {r} // one of enumerated values(USA, Canada,
Mexico, Unknown)
```

Country code of the issuing bank, parsed from the MICR data. One of the OPOS constants MICR\_CC\_USA, MICR\_CC\_CANADA, MICR\_CC\_MEXICO, or MICR\_CC\_UNKNOWN.

### **IMICRPlugin.EPC**

```
property EPC : Integer; {r} // extended process code
```

Extended process code parsed from the MICR data.

### **IMICRPlugin.SerialNumber**

```
property SerialNumber : String; {r}
```

The check serial number parsed from the MICR data.

### **IMICRPlugin.TransitNumber**

```
property TransitNumber : String {r}
```

The transit number, parsed in from the MICR data.

## **IPinPadPlugin**

### **IPINPadPlugin.OnPinPadDataEvent**

```
Property OnPinPadDataEvent;
```

Set by RetailPro to an event handler address. When the PIN pad device signals a data event, the Plugin should pass that signal on to RetailPro.

## **IScalePlugin**

### **IScalePlugin.DisplayText**

```
function DisplayText(AText : String) : Integer;
```

This instructs the Plugin to display the specified text on the scale's display, if it has one.

### **IScalePlugin.ReadWeight**

```
function ReadWeight : Integer;
```

This returns the weight as an integer with an assumed decimal in the thousands place.

### **IScalePlugin.ZeroScale**

```
function ZeroScale : Integer;
```

This instructs the Plugin to tell the scale device to zero itself, if it has that capability.

## **IBarcodeScannerPlugin**

### **IBarcodeScannerPlugin.ScanData**

```
property ScanData: PChar read Get_ScanData;
```

After a DataEvent is signaled, ScanData contains the string captured from the bar code scanning device.

## **IIInventoryScannerPlugin**

### **IIInventoryScannerPlugin.UploadData**

```
function UploadData(ALookupType: Integer): Integer; safecall;
```

UploadData is used to tell the Plugin that RetailPro has prepared data to be transmitted to the inventory scanner. The Plugin is expected to read the drop file created by RetailPro and push the data up to the inventory scanner device. See the inventory scanner documentation for more details.

### **IIInventoryScannerPlugin.DownloadData**

```
function DownloadData(ALookupType: Integer): Integer; safecall;
```

DownloadData is used to tell the Plugin to create the data that RetailPro will be inputting. Upon return, RetailPro expects a drop file with a specific file layout that it can import into its database. See the inventory scanner documentation for more details.

## **IFiscalPrinterPlugin**

### **About the IFiscalPrinterPlugin**

In some locales, governments require retailers to use an approved fiscal printer that records specific information (to improve efficiency of tax collection).

The IFiscalPrinter interface in the Retail Pro 9 Plugin Types Library enables development partners to create Plugins that RP9 uses to transmit fiscal information to a fiscal printer. The methods and properties defined on the IFiscalPrinter interface are derived from the OPOS IOPOSFiscalPrinter interface. Some of the method names are not identical, but there is a one-to-one equivalence.

Please note that only a small number of the methods and properties defined in this interface are used by Retail Pro 9. The unused methods in the IFiscalPrinter interface were included because few fiscal printers operate identically or share the same fiscal requirements. When the interface was defined, all the possible methods and properties were included. Support for the various methods and properties is coded into RP9 as needed.

### ***IFiscalPrinter Methods, Properties Used by RP9***

#### **Properties used by RP9**

- AdditionalInfo1, AdditionalInfo2
- CustID, CustName, CustPhone, CustAddress
- DayOpened
- POSID
- DescriptionId

#### **Methods used by RP9**

- AddItemAdjustment
- AddItem
- StartReceipt
- EndReceipt
- PrintTotal
- StartRefundReturn
- StartDay
- PrintXReport
- PrintZReport
- Reset

#### **IFiscalPrinterPlugin.AddItem**

```
function AddItem(ADescription: PChar; APrice: Currency; AQuantity: Integer;  
AVatInfo: Integer; AUnitPrice: Currency; AUnitName: PChar): Integer;
```

This tells the Plugin to add an item to the current document. This must be preceded with a call to StartReceipt to put the device in fiscal document mode.

- ADescription: A text description, 40 characters maximum, of the line item.
- APrice: The extended price for this detail.
- AQuantity: The number of items.
- AVatInfo: VAT tax information. This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.
- AUnitPrice: The price of single item.
- AUnitName: Unit by which the item is enumerated, e.g. "pc.", "oz.", "lbs.", etc.

### **IFiscalPrinterPlugin.AddItemAdjustment**

```
function AddItemAdjustment(AAdjustmentType: Integer; ADescription:PChar;  
AAmount: Currency; AVatInfo: Integer): Integer;
```

This tells the Plugin to apply a single line adjustment, as in a discount or fee. This must be preceded with a call to StartReceipt to put the device in fiscal document mode.

- AAdjustmentType: One of the enumerated constants in FiscalPrinterAdjustmentTypes.
- ADescription: A text description, 40 characters maximum, of the line item.
- AAmount: Amount of adjustment.
- AVatInfo: VAT tax information. This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.

### **IFiscalPrinterPlugin.StartDay**

```
function StartDay(ADocumentAmount: Integer): Integer;
```

This instructs the Plugin to tell the device to begin a fiscal day. Fiscal printers, as a convention, do not limit the number of fiscal days that can be run in a single real day, so this method, along with EndDay, can be called multiple times within a calendar day.

ADocumentAmount: Intended for future development, and should be set to zero for now.

### **IFiscalPrinterPlugin.StartReceipt**

```
function StartReceipt(APrintHeader: WordBool): Integer;
```

This instructs the Plugin to put the device in fiscal document mode. This typically causes the device to print all header lines and, while in this mode, allows the use of methods that describe the content of the receipt. EndReceipt finalizes the fiscal document.

### **IFiscalPrinterPlugin.EndReceipt**

```
function EndReceipt: Integer;
```

Tells the Plugin to signal the device to end the receipt. This typically causes the device to print totals and any defined trailer and footer lines. This also puts the device in a non-document mode.

### **IFiscalPrinterPlugin.PrintTotal**

```
function PrintTotal(AAmount: Currency; APayment: Currency; ADescription: PChar): Integer;
```

Tells the Plugin to signal the device to print the outstanding total of the current document, the amount tendered for payment, and an optional description of the total line. Note that the amount does not have to match the payment. For instance, the parameters could be set to AAmount=212.00, APayment=20.00, and ADescription="Coupon". In that case, the entire document hasn't been paid yet, so a call to EndReceipt after this call would be rejected. The entire amount of the document must be accounted for prior to calling EndReceipt.

- AAmount: The outstanding total amount to print.
- APayment: The amount tendered.
- ADescription: An optional description of the amount and tendered amount.

### **IFiscalPrinterPlugin.StartRefundReturn**

```
function StartRefundReturn(ACompanyName: PChar; ACustomerID: PChar; ASaleDate: TDateTime; AInvoiceNo: PChar; ARegisterNo: PChar): Integer;
```

This method is called when printing an invoice with an invoice type of "return". This puts the device in fiscal document mode and enables calls that add details to the receipt.

- ACompanyName: The company, if applicable, on the receipt.
- ACustomerID: If one was included on the receipt, this is set to the bill to customer SID.
- ASaleDate: The date of sale on the receipt.
- AInvoiceNo: The invoice number.
- ARegisterNo: The workstation ID that the receipt was printed on.

### **IFiscalPrinterPlugin.PrintXReport**

```
function PrintXReport: Integer;
```

This tells the Plugin to signal the device to generate an 'X-Out' report. This can be called multiples time during the day.

### **IFiscalPrinterPlugin.PrintZReport**

```
function PrintZReport: Integer;
```

This tells the Plugin to signal the device to generate an 'Z-Out' report. This report must be run at least once a day.

### **IFiscalPrinterPlugin.Reset**

```
function Reset: Integer;
```

This tells the Plugin to reset the print device. A reset should cancel any document mode, reset any special settings, and leave the device ready to resume normal operation.

## **Methods NOT Currently Used by RP9**

The following methods are defined for future use.

### **IFiscalPrinterPlugin.Capability (future release)**

```
function Capability(ACapability: Integer): WordBool;
```

When passed one of the FiscalPrinterCaps enumerated constants, the Plugin must return the appropriate Boolean value to indicate whether the physical device is capable of that functionality.

### **IFiscalPrinterPlugin.AddFuelSurcharge (future release)**

```
function AddFuelSurcharge(ADescription: PChar; APrice: Currency; AQuantity: Integer; AVatInfo: Integer; AUnitPrice: Currency; AUnitName: PChar; ASpecialTax: Integer; ASpecialTaxName: PChar): Integer;
```

This adds a fuel charge fee to a receipt. This must be preceded with a call to StartReceipt to put the device in fiscal document mode.

- ADescription: A text description, 40 characters maximum, of the line item.
- APrice: The extended price for this detail.
- AQuantity: The number of items.
- AVatInfo: VAT tax information. The entry in the printer's VAT table. Default = 0.
- AUnitPrice: The price of single item.
- AUnitName: Name of units by which the item is enumerated (e.g. "pc.", "oz.", "lbs.")
- ASpecialTax: Additional tax amount.
- ASpecialTaxName: Description of additional tax.

### **IFiscalPrinterPlugin.AddNotPaid (future release)**

```
function AddNotPaid(ADescription: PChar; AAmount: Currency): Integer;
```

Intended for future implementation, this is intended to display an unpaid portion of the total amount of the receipt. This must be preceded with a call to StartReceipt to put the device in fiscal document mode.

- ADescription: A text description, 40 characters maximum, of the line item.
- AAmount: The amount of the outstanding portion of the receipt.

### **IFiscalPrinterPlugin.AddItemPackageAdjustment (future release)**

```
function AddPackageAdjustment(AAdjustmentType: Integer; ADescription: PChar; AVatAdjustment: PChar): Integer;
```

This is used to apply an adjustment to a package, kit, or similar grouping of items.

- AAdjustmentType: One of the enumerated constants in FiscalPrinterAdjustmentTypes.
- ADescription: A text description, 40 characters maximum, of the line item.
- AVatAdjustment: Description, limited to 40 chars, of any effect on VAT.

### **IFiscalPrinterPlugin.AddRefund (future release)**

```
function AddRefund(ADescription: PChar; AAmount: Currency; AVatInfo: Integer): Integer;
```

This tells the Plugin to add a specific line item refund.

- ADescription: A text description, 40 characters maximum, of the line item.
- AAmount: The amount of the line item refund.
- AVatInfo: VAT tax information. This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.

### **IFiscalPrinterPlugin.ClearPrinterError (future release)**

```
function ClearPrinterError: Integer;
```

Tells the Plugin to reset all the error condition properties, codes, and descriptions.

### **IFiscalPrinterPlugin.EndDay (future release)**

```
function EndDay: Integer;
```

Tells the Plugin to signal the device to perform an end-of-day operation.

### **IFiscalPrinterPlugin.EndSlip (future release)**

```
function EndSlip(ATimeout: Integer): Integer;
```

Tells the Plugin to signal the device that the slip is complete and to put the device back in a non-document mode.

ATimeout: Seconds after which, if the device does not complete the slip, the device should signal a timeout. If the device is not capable of signaling a timeout, the Plugin should provide that capability.

### **IFiscalPrinterPlugin.PrintReport (future release)**

```
function PrintReport(AReportType: Integer; AStartNum: PChar; AEndNum: PChar): Integer;
```

Tells the printer to signal the device to print the indicated report type.

- AReportType: One of...
- AStartNum: Starting transaction number (if supported).
- AEndNum: Ending transaction number (if supported).

### **IFiscalPrinterPlugin.PrintSlipLine (future release)**

```
function PrintSlipLine(ADocumentLine: PChar): Integer;
```

Tells the printer to send a line of text to the device. This call must be preceded by a call to StartSlip.

ADocumentLine: Text to print.

### **IFiscalPrinterPlugin.PrintSubtotal (future release)**

```
function PrintSubtotal(AAmount: Currency): Integer;
```

Subtotal the line item charges, refunds and adjustments up to this point and print that amount. The print device must be in a document mode prior to making this call.

AAmount: The amount of the subtotal.

**IFiscalPrinterPlugin.PrintTaxID (future release)**

```
function PrintTaxID(ATaxID: PChar): Integer;
```

Prints the tax ID.

ATaxID: The ID for the tax to print.

**IFiscalPrinterPlugin.PrintTotal (future release)**

```
function PrintTotal(AAmount: Currency; APayment: Currency; ADescription: PChar): Integer;
```

Tells the Plugin to signal the device to print the outstanding total of the current document, the amount tendered for payment, and an optional description of the total line. Note that the amount does not have to match the payment. For instance, the parameters could be set to AAmount=212.00, APayment=20.00, and ADescription="Coupon". In that case, the entire document hasn't been paid yet, and so a call to EndReceipt after this call would be rejected. The entire amount of the document must be accounted for prior to calling EndReceipt.

- AAmount: The outstanding total amount to print.
- APayment: The amount tendered.
- ADescription: An optional description of the amount and tendered amount.

**IFiscalPrinterPlugin.PrintYReport (future release)**

```
function PrintYReport: Integer;
```

This tells the Plugin to signal the device to generate an 'X' report.

**IFiscalPrinterPlugin.VoidItem (future release)**

```
function VoidItem(ADescription: PChar; APrice: Currency; AQuantity: Integer; AVatInfo: Integer; AUnitPrice: Currency; AUnitName: PChar): Integer;
```

This is not currently called by RetailPro, but is included for future use. This causes the device to print a void for a single line item. Note that the price and quantity coming from RetailPro will be positive.

- ADescription: A text description, 40 characters maximum, of the line item.
- APrice: The extended price for this detail.
- AQuantity: The number of items.
- AVatInfo: VAT tax information. This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.
- AUnitPrice: The price of single item.
- AUnitName: The name of the units by which the item is enumerated, as in "pc.", "oz.", "lbs.", etc.



### **IFiscalPrinterPlugin.VoidItemAdjustment (future release)**

```
function VoidItemAdjustment(AAdjustmentType: Integer; ADescription:PChar;  
AAmount: Currency; AVatInfo: Integer): Integer;
```

This is not currently called by RetailPro, but is included for future use. This will tell the Plugin to send a line item void of an adjustment.

- AAdjustmentType: One of the enumerated constants in FiscalPrinterAdjustmentTypes.
- ADescription: A text description, 40 characters maximum, of the line item.
- AAmount: The outstanding total amount to print.
- AVatInfo: VAT tax information. This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.

### **IFiscalPrinterPlugin.VoidFuelSurcharge (future release)**

```
function VoidFuelSurcharge(ADescription: PChar; APrice: Currency; AVatInfo:  
Integer; ASpecialTax: Currency): Integer;
```

This call is intended to void a single line fuel surcharge previously added to the receipt.

- ADescription: A text description, 40 characters maximum, of the line item.
- APrice: The extended price for this detail.
- AVatInfo: VAT tax information. This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.
- ASpecialTax: Any tax amount included in this surcharge.

### **IFiscalPrinterPlugin.VoidPackageAdjustment (future release)**

```
function VoidPackageAdjustment(AAdjustmentType: Integer; AVatAdjustment:  
PChar): Integer;
```

This is intended to void a line adjustment to a package, kit, or similar item grouping. This is not currently used, and is included for future use.

- AAdjustmentType: One of the enumerated constants in FiscalPrinterAdjustmentTypes.
- AVatAdjustment: Documentation pending

### **IFiscalPrinterPlugin.VoidRefund (future release)**

```
function VoidRefund(ADescription: PChar; AAmount: Currency; AVatInfo: Integer):  
Integer;
```

This is not currently used, and is include for future use. This is intended to void a single line refund.

- ADescription: A text description, 40 characters maximum, of the line item.
- AAmount: The amount of the refund.
- AVatInfo: This number refers to the VAT tax entry in the printer's VAT table, if applicable. Default is zero.

### **IFiscalPrinterPlugin.Reset (future release)**

```
function Reset: Integer;
```

This tells the Plugin to reset the print device. A reset should cancel any document mode, reset any special settings, and leave the device ready to resume normal operation.

### **IFiscalPrinterPlugin.StartDay**

```
function StartDay(ADocumentAmount: Integer): Integer;
```

This instructs the Plugin to tell the device to begin a fiscal day. Fiscal printers, as a convention, do not limit the number of fiscal days that can be run in a single real day, so this method, along with EndDay, can be called multiple times within a calendar day.

ADocumentAmount: This is intended for future development, and should be set to zero for now.

### **IFiscalPrinterPlugin.StartReceipt**

```
function StartReceipt(APrintHeader: WordBool): Integer;
```

This instructs the Plugin to put the device in fiscal document mode. This typically causes the device to print all header lines and, while in this mode, allows the use of methods that describe the content of the receipt. EndReceipt finalizes the fiscal document.

### **IFiscalPrinterPlugin.StartSlip (future release)**

```
function StartSlip(ATimeout: Integer): Integer;
```

This instructs the Plugin to put the device in non-fiscal document mode. This allows the use of methods for defining the content of the slip. EndSlip finalizes the non-fiscal document.

### **IFiscalPrinterPlugin.VoidReceipt (future release)**

```
procedure VoidReceipt(var Value: Integer);
```

This tells the Plugin to instruct the device to void an entire receipt amount.

Value: The total value of the receipt being voided.

### **IFiscalPrinterPlugin.ActualCurrency (future release)**

```
property ActualCurrency: Integer readonly
```

This property defines the type of currency being accepted. Valid values are defined in the enumerated type FiscalPrinterCurrencyIDs.

### **IFiscalPrinterPlugin.AdditionalHeader (future release)**

```
property AdditionalHeader: PChar
```

When a call to StartReceipt is made, this property defines the additional header line to be printed on the receipt, if that functionality is supported by the device and/or Plugin.

### **IFiscalPrinterPlugin.AdditionalTrailer (future release)**

```
property AdditionalTrailer: PChar
```

When finalizing a receipt using EndReceipt, this property defines the additional trailer line to be printed at the end of the receipt.

### **IFiscalPrinterPlugin.AmountDecimalPlaces (future release)**

property AmountDecimalPlaces: Integer readonly

This property defines how many decimal places should be assumed when processing amount fields.

### **IFiscalPrinterPlugin.ChangeDueLabel (future release)**

property ChangeDueLabel: PChar

This property defines the label to be used when printing lines describing the change due on the receipt, if the device supports that capability.

### **IFiscalPrinterPlugin.CheckTotal (future release)**

property CheckTotal: WordBool

This property, when set to true, enables the ability to validate the total amount specified by the application for the receipt against the device's tally.

### **IFiscalPrinterPlugin.ContractorID (future release)**

property ContractorID: Integer

This property defines the identification of the contractor to whom the receipt or some portion of the receipt items are assigned. On some types of devices, this can affect the header lines and totalizers used.

### **IFiscalPrinterPlugin.CountryCode (future release)**

property CountryCode: Integer readonly

Where supported, this can be used to query which countries the fiscal printer device supports. One of the enumerated constants in FiscalPrinterCountryCodes.

### **IFiscalPrinterPlugin.CoverOpen (future release)**

property CoverOpen: WordBool readonly

Where supported, this property indicates whether the cover on the printer is open or not.

### **IFiscalPrinterPlugin.ConfigurationDate (future release)**

property ConfigurationDate: PChar

Where supported, this is the date the printer was last configured.

### **IFiscalPrinterPlugin.LastEndOfDayDate (future release)**

property LastEndOfDayDate: PChar

Where supported, this is the date the last end-of-day was performed.

### **IFiscalPrinterPlugin.LastResetDate (future release)**

property LastResetDate: PChar

Where supported, this is the date the last reset was performed on the print device.

### **IFiscalPrinterPlugin.RealTimeClock (future release)**

property RealTimeClock: PChar

Where supported, this is the value currently stored in the print device's real time clock.

### **IFiscalPrinterPlugin.LastVATChange (future release)**

property LastVATChange: PChar

Where supported, this is the date the VAT table in the fiscal printer was last updated.

### **IFiscalPrinterPlugin.LastFiscalDayStartDate (future release)**

property LastFiscalDayStartDate: PChar

Where supported, this is the date that the last fiscal day was begun.

### **IFiscalPrinterPlugin.DayOpened (future release)**

property DayOpened: WordBool readonly

If this property is true, a fiscal day has been started and not yet finalized, and calls to StartReceipt and StartSlip will succeed.

### **IFiscalPrinterPlugin.DescriptionLength (future release)**

property DescriptionLength: Integer readonly

This property should return the maximum description length supported by the physical device.

### **IFiscalPrinterPlugin.DuplicateReceipt (future release)**

property DuplicateReceipt: WordBool

When set to true, this indicates that the receipt being printed is a duplicate. The buckets in the device are not affected during this time.

### **IFiscalPrinterPlugin.FiscalErrorLevel (future release)**

property FiscalErrorLevel: Integer readonly

Indicates the severity of the last error that occurred. One of the enumerated constants in FiscalPrinterErrorLevels.

### **IFiscalPrinterPlugin.FiscalErrorOutID (future release)**

property FiscalErrorOutID: Integer readonly

This is set to the error ID for the last error that occurred. One of the enumerated constants in FiscalPrinterExtendedErrorCodes.

### **IFiscalPrinterPlugin.FiscalErrorState (future release)**

property FiscalErrorState: Integer readonly

This indicates the state the physical device was in when the last error occurred. One of the enumerated constants in FiscalPrinterStates.

### **IFiscalPrinterPlugin.FiscalErrorStation (future release)**

property FiscalErrorStation: Integer readonly

This indicates which station (receipt=0, slip=1) that the printer was printing to when the last error occurred.

### **IFiscalPrinterPlugin.FiscalErrorString (future release)**

property FiscalErrorString: PChar readonly

This is the text message describing the last error that occurred.

### **IFiscalPrinterPlugin.FiscalReceiptStation (future release)**

property FiscalReceiptStation: Integer

One of the enumerated constants in FiscalPrinterStations.

### **IFiscalPrinterPlugin.FiscalReceiptType (future release)**

property FiscalReceiptType: Integer

One of the enumerated constants in FiscalPrinterReceiptType.

### **IFiscalPrinterPlugin.FlagWhenIdle (future release)**

property FlagWhenIdle: WordBool

Documentation pending.

### **IFiscalPrinterPlugin.JournalEmpty (future release)**

property JournalEmpty: WordBool readonly

This is not currently called by RetailPro, but is included for future use. Where supported this flag indicates whether the journal tape reel is empty, if the device has the capability to report this. When True, the tape has run out and must be changed. If not supported by the device, this property should always return False.

### **IFiscalPrinterPlugin.JournalNearEnd (future release)**

property JournalNearEnd: WordBool readonly

This is not currently called by RetailPro, but is included for future use. Where supported this flag indicates whether the journal tape reel is near empty, if the device has the capability to report this. When True, the tape should be changed prior to creating another receipt. If not supported by the device, this property should always return False.

### **IFiscalPrinterPlugin.MaxMessageLength (future release)**

property MaxMessageLength: Integer readonly

Where supported, this should return the maximum width a string message. This usually equates the maximum number of characters that can be printed on a single line.

### **IFiscalPrinterPlugin.PrintNormal (future release)**

```
function PrintNormal(AStation: Integer; AMessageType: Integer; AData: PChar): Integer;
```

This is used to send application defined text to the printer.

- AStation: One of the enumerated constants in FiscalPrinterStations.
- AMessageType: One of the enumerated constants in FiscalPrinterMessageType;
- AData: The text string to print.

### **IFiscalPrinterPlugin.MaxHeaderLines (future release)**

property MaxHeaderLines: Integer readonly

This is used to indicate the maximum number of header lines allowed on a receipt.

### **IFiscalPrinterPlugin.MaxTrailerLines (future release)**

property MaxTrailerLines: Integer readonly

This indicates the maximum number of trailer, or footer, lines allowed on a receipt.

### **IFiscalPrinterPlugin.MaxVATRates (future release)**

property MaxVatRates: Integer readonly

This indicates the maximum number of entries allowed in the device's internal VAT rates table.

### **IFiscalPrinterPlugin.PostLine (future release)**

property PostLine: PChar

This is not currently called by RetailPro, but is included for future use. Where supported, this property holds an extra line to be printed following the next detail that's printed on the receipt.

### **IFiscalPrinterPlugin.PredefinedPaymentLines (future release)**

property PredefinedPaymentLines: PChar readonly

Documentation pending.

### **IFiscalPrinterPlugin.PreLine (future release)**

property PreLine: PChar

This is not currently called by RetailPro, but is included for future use. Where supported, this property holds an extra line to be printed just prior to the next detail that's printed on the receipt.

### **IFiscalPrinterPlugin.PrinterState (future release)**

property PrinterState: Integer readonly

This indicates the state or mode that the printer is currently in. Set to one of the enumerated constants in FiscalPrinterStates.

### **IFiscalPrinterPlugin.QuantityDecimalPlaces (future release)**

property QuantityDecimalPlaces: Integer readonly

This indicates the number of decimals places that the fiscal printer assumes when passed an integer amount value.

### **IFiscalPrinterPlugin.MaxQuantityLength (future release)**

property MaxQuantityLength: Integer readonly

This indicates the maximum number of characters that can be passed in to express the quantity of a given item on a single detail line.

### **IFiscalPrinterPlugin.ReceiptEmpty (future release)**

property ReceiptEmpty: WordBool readonly

This is not currently called by RetailPro, but is included for future use. Where supported, this indicates that the feeder spool for receipts is empty and must be changed out to continue processing the receipt.

### **IFiscalPrinterPlugin.ReceiptNearEnd (future release)**

property ReceiptNearEnd: WordBool readonly

This is not currently called by RetailPro, but is included for future use. Where supported, this indicates that the feeder spool for receipts is nearly empty and should be changed out prior to creating a new receipt.

### **IFiscalPrinterPlugin.RemainingFiscalMemory (future release)**

property RemainingFiscalMemory: Integer readonly

This is not currently called by RetailPro, but is included for future use. Where supported, this indicates the remaining memory in the fiscal printer for transaction data.

### **IFiscalPrinterPlugin.ReservedWord (future release)**

property ReservedWord: PChar readonly

This property contains the word automatically printed on the receipt when totals are printed.

### **IFiscalPrinterPlugin.SlipEmpty (future release)**

property SlipEmpty: WordBool readonly

This is not currently called by RetailPro, but is included for future use. Where supported, this indicates that the feeder spool for slip is empty and must be changed out to continue processing a slip..

### **IFiscalPrinterPlugin.SlipNearEnd**

property SlipNearEnd: WordBool readonly

This is not currently called by RetailPro, but is included for future use. Where supported, this indicates that the feeder spool for slips is nearly empty and should be changed out soon.

### **IFiscalPrinterPlugin.SlipSelection (future release)**

property SlipSelection: Integer

This is not currently called by RetailPro, but is included for future use. This indicates the type of document to print through the slip station. One of the enumerated constants in FiscalPrinterSlipSelections.

### **IFiscalPrinterPlugin.TotalizerType (future release)**

property TotalizerType: Integer

This is not currently called by RetailPro, but is included for future use. This indicates the type of totalizer to use for subsequent details. One of the enumerated constants in FiscalPrinterTotalizerTypes.

### **IFiscalPrinterPlugin.TrainingMode (future release)**

property TrainingMode: WordBool

Where supported, this puts the device into "training mode." All transactions sent to the printer are printed and tallied, and reports can be generated against that data, but when training mode is exited, the data is deleted.

### **IFiscalPrinterPlugin.PrintPeriodicTotalsReport (future release)**

```
function PrintPeriodicTotalsReport(BeginDate: PChar; EndDate: PChar): Integer;
```

This is not currently called by RetailPro, but is included for future use. Where supported, this prints a report of totals between the dates specified.

BeginDate: Start of date range, in ddmmyyhhnn format.

EndDate: End of date range, in ddmmyyhhnn format.

### **IFiscalPrinterPlugin.PrintPowerLossReport (future release)**

```
function PrintPowerLossReport: Integer;
```

This is not currently called by RetailPro, but is included for future use. Where supported, this prints a report of a power failure that resulted in a loss of data stored in the fiscal printer memory.

### **IFiscalPrinterPlugin.PrintReceiptCashInOut (future release)**

```
function PrintReceiptCashInOut(AAmount: Currency): Integer;
```

This is not currently called by Retail Pro but is included for future use. Where supported, this prints a cash-in or cash-out receipt amount on the receipt.

### **IFiscalPrinterPlugin.PrintDuplicateReceipt (future release)**

```
function PrintDuplicateReceipt: Integer;
```

This is not currently called by RetailPro, but is included for future use. Where supported, this prints a duplicate of a buffered transaction.

### **IFiscalPrinterPlugin.FiscalErrorExtended (future release)**

```
property FiscalErrorExtended: Integer readonly
```

This contains the extended (more specific) error code for the last error code that occurred, if applicable. One of the enumerated constants defined in FiscalPrinterExtendedErrorCodes.

### **IFiscalPrinterPlugin.POSID (future release)**

```
property POSID: PChar
```

This property is set to a unique identifier for the workstation. In this case, the workstation name is passed in.

### **IFiscalPrinterPlugin.CashierID (future release)**

```
property CashierID: PChar
```

This property is set to the cashier's unique employee ID.

### **IFiscalPrinterPlugin.StoreID (future release)**

```
property StoreID: PChar
```

This property is set to a combination of the subsidiary number and store number to uniquely identify the store.

### **IFiscalPrinterPlugin.EndRefundReturn (future release)**

```
function EndRefundReturn: Integer;
```

This ends the current return receipt, prints totals, and takes the device out of fiscal document mode.



## User-Interface Data Access

When a user interface Plugin has been triggered, the Plugin has access to the business objects via methods on the adapter, as well as being able to request access to referenced business objects (business objects used to pull related information from other datasets) and even to request that new business objects be created.

All business objects are accessed via handles, and attributes on those business objects are accessed by name. Simple preference settings are available by ID using the enumerated type BOPreference.

Direct access to Retail Pro data is available by submitting SQL statements. Result sets are returned in variant arrays.

If a Plugin needs to persist information, it can either do so using its own data storage mechanism or by submitting SQL statements that make calls to predefined stored procedures that give the Plugin access to a set of tables in the Retail Pro tablespace.

*Note:* It is assumed that Plugins may need to communicate directly with each other, and that it may be advantageous for them to see the same information. To that end, business object handles can be cloned.

### Business Object Data Access

Data in RetailPro should be accessed via the BOGetAttributeValueByName and BOSetAttributeValueByName methods. These are as follows:

```
function BOSetAttributeValueByName(BOHandle: Integer; AttrName: PChar; AValue: OleVariant): Integer; safecall;
```

BOHandle	The integer handle used to access the parent and child business objects. Zero indicates the parent business object.
AttrName	The name of the attribute being accessed.
AValue	The value to store in the attribute
Returns	An integer indicating the status of the operation. See "Enumerated Values – Error Codes"

```
function BOGetAttributeValueByName(BOHandle: Integer; AttrName: PChar): OleVariant; safecall;
```

BOHandle	The integer handle used to access the parent and child business objects. Zero indicates the parent business object.
AttrName	The name of the attribute being accessed.
Returns	The value of the attribute. Can be Null.

## **Direct Data Access**

Plugin writers can use SQL statements against the database. By providing indirect access to a separate database session against which a special Plugin database role has been granted, we can allow the Plugin the following types of access:

- Read access on nearly all Retail Pro 9 data
- Create/Read/Write/Delete/Drop access on any tables owned by the Plugin
- Access to specific "safe" Retail Pro 9 stored procedures
- Create/Execute access to stored procedures owned by the Plugin

By using a user role to control access, we can ensure that:

- The Plugin does not harm any of the Retail Pro 9-owned data
- Store procedures owned by the Plugin cannot harm any Retail Pro 9-owned data
- The Plugin has full capabilities for manipulating its own data

*Note:* Retail Pro 9 will necessarily need to be granted access to all Plugin data

## Plugins Type Library (Plugins.tlb)

The Plugins type library consists of:

- User Interfaces
- Parent Interfaces
- Base Interfaces
- Supported Procedures
- Classes
- Enumerated Values
- Preference Constants

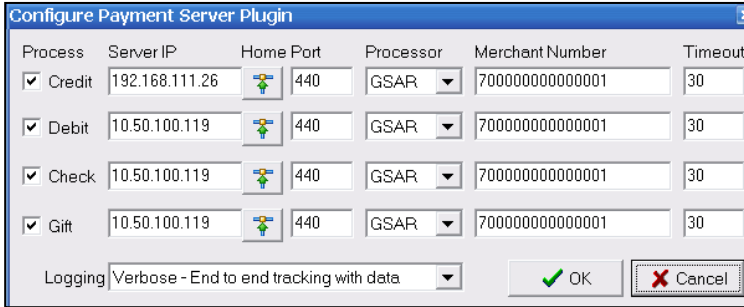
### Interfaces

The interfaces defined in the Plugins type library are divided into two types:

- User interfaces
- Hardware interfaces

#### *User Interfaces*

<i>Interface</i>	<i>Notes</i>
IPluginAdapter	Used to define the Adapter reference for each Plugin. Enables the Plugin to communicate with RetailPro, to create and use business objects, to read Retail Pro's configuration settings, and signal RetailPro of events.
IDiscover	At least one class in the COM server module must implement this interface.

Interface	Notes
IConfigure	<p>Allows the Retail Pro user to access to configure the settings specific to the Plugin. For each Plugin that implements IConfigure, the description for the Plugin server specified by its IDiscover Plugin is displayed in Retail Pro's "Workstation Preferences section." Double-clicking the workstation preferences entry for the Plugin server fires the IConfigure's HandleEvent method. It is the Plugin's responsibility to display whatever forms are required, handle all configuration editing tasks within that form, and manage the appropriate storage of the configuration settings. (See sample below.)</p> <p><i>Sample Plugin Configuration Screen:</i></p> 
IAttributeValidationPlugin	Intercepts values before they are assigned to an attribute, and can stop further validation processing and acceptance of the value.
IAttributeAssignmentPlugin	Is invoked after the user assigns a new value to a field.
IEntityUpdatePlugin	Is called before and after a business object's data is stored in the database, or when editing on the business object is cancelled.
ISideButtonPlugin	Is used to add a custom button to a context (side) menu.
IPrintPlugin	Is given access to the XML structure used to print documents for a given business object type just prior to the process that uses the XML structure to send data to the printer.
ITenderPlugin	Is called when the tender screen is entered or tender information is changed.
ICustomAttributePlugin	Add a custom field to a Retail Pro screen.

<i>Interface</i>	<i>Notes</i>
IEFTPlugin	Handles requests to process authorizations for credit card, debit card, gift card, and check tenders. It is provided with all the information from the user and any POS hardware and is expected to return authorization and/or error codes from the EFT processor.
IItemAddRemovePlugin	Called whenever a detail item is added or removed from a document, e.g., Invoice and invoice items.
IServerPlugin	Activated when Retail Pro starts up and destroyed when RetailPro shuts down. It is not triggered by Retail Pro, but has the ability to see Retail Pro's data, create BOs and use them, execute SQL queries, etc.
ILoginPlugin	Instantiated whenever a user logs in or out, is signaled that a login or logout has occurred, and then freed. When it's called during a login, it can abort that login.
ITenderDialoguePlugin	Used to completely replace the data entry and, if appropriate, EFT authorization, of a tender entry.

### **Hardware Interfaces**

<i>Interface</i>	<i>Notes</i>
IDisplayPlugin	Line display device.
ICashDrawerPlugin	Cash drawer device.
IMSRPlugin	Magnetic Stripe Reader (MSR) device.
IPinPadPlugin	Pin pad device.
IScalePlugin	Scale or other weight input device.
IMICRPlugin	Magnetic Ink Character Recognition scanner.
ICheckImageScannerPlugin	Check image scanning device.
IBarCodeScannerPlugin	Bar code scanning device.
ISigCapPlugin	Signature capture device.
IInventoryScannerPlugin	Hand held inventory maintenance device.
IFiscalPrinterPlugin	Fiscal printing device.

# The Plugin Adapter Interface

This is used to define the Adapter reference for each Plugin. It provides a means for the Plugin to communicate with RetailPro, create and use business objects, access business objects referenced by attributes, read RetailPro's configuration settings, signal RetailPro of events, and provides a SQL session for accessing the Retail Pro tablespace directly.

## IPluginAdapter

### *AllBONames*

#### Parameters

None

#### Returns

OLEVariant array of strings

#### Description

This returns a list of all the business objects supported by the PI-API. These names can be used by CreateChildBOByName to instantiate new business objects.

### *APIVersion*

#### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
MajorVersion	Integer	R	(see common description for BOHandle)
MinorVersion	Integer	R	(see common description for variant arrays)
Revision	Integer	R	(see common description for variant array)

#### Returns

All three parameters are "out" parameters. Values are supplied in the parameter names.

#### Description

This method returns the highest version of the PI-API that Retail Pro supports, in the form of three integer values representing the major, minor and revision numbers. For instance, it might return MajorVersion=2, MinorVersion=1, and Revision=1, to indicate the 1<sup>st</sup> revision of the PI-API interface version 2.1.

## PluginCapability

### Parameters

Param	Type	Access	Description
ACapability	Integer	W	Capability constant

### Returns

True if Plugin supports ACapability. Typically False.

### Description

This method is designed for forward compatibility of the PI-API. Returning false will always allow for the Plugin functionality designed as of version 1.2 of the PI-API. If a particular interface is extended, the PluginManager will query each Plugin using this interface to see if it supports the new functionality by passing an ACapability constant. If the Plugin returns True, the PluginManager will allow RetailPro to call any new extended methods of this interface. It is up to the Plugin to decide if it supports this capability. A Plugin that returns True that does NOT support the new functionality will cause an access violation and be disabled.

## BOUpdateDataSetRecords

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
IncludeRefs	Boolean	W	If set to True (-1), this tells the business object to also update any reference business objects associated with attributes on the specified business object that are in edit or insert mode, and is propagated to those business objects' reference BOs.

### Returns

Integer status code. Possible values:

peSuccess

peInvalidBOHandle

peUnableToUpdateDatasetRecords

### Description

This method tells the business object to post all changes to its attributes to the database. If IncludeRefs is set to True (-1), the call is propagated to all reference business objects. A good example of this might be an update to an invoice. By setting IncludeRefs to True, the update operation would include the invoice details, customer information, etc.

This routine is used with business objects that use datasets. There are some business objects that are used as temporary data holders or are persisted manually rather than being directly connected to that database; this routine does not affect those business objects.

Note that the routine called by this method is also triggered by calling BOPost, but calling this routine skips triggering of pre- and post-update logic. The preferable method to use is BOPost.

### ***BOSetAttributeValueByName***

#### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AttrName	PChar	W	(see common description for AttrName)

#### **Returns**

OLEVariant containing the value of the attribute specified.

#### **Description**

Given the handle to a business object and the name of an attribute in that business object, this sets an OLEVariant with the value of that attribute. See common description of attribute value variants.

### ***BOGetAttributeValueByName***

#### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AttrName	PChar	W	(see common description for AttrName)

#### **Returns**

OLEVariant containing the value of the attribute specified.

#### **Description**

Given the handle to a business object and the name of an attribute in that business object, this returns an OLEVariant with the value of that attribute. See common description of attribute value variants.



## **BOSetAttributeValues**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeNames	OLEVariant	W	(see common description for variant arrays)
AValue	OLEVariant	W	(see common description for variant arrays)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToSetAttributeValues

### **Description**

Given the handle to a business object, an OLEVariant array of names of attributes in that business object, and an OLEVariant array of OLEVariants containing the values for that attribute, this sets the attributes to the values specified. The business object must be in insert or edit mode before setting attribute values. This allows your Plugin to write large numbers of values in one call through the PI-API.

## **BOGetAttributeValues**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeNames	OLEVariant	W	(see common description for variant arrays)

### **Returns**

Variant array of variant attribute values. (see common description for variant arrays)

### **Description**

Given the handle to a business object and a variant array of names of attributes in that business object, this returns an OLEVariant array of OLEVariants containing the values of the attributes. While there is some overhead in constructing the array and processing it once it's received, this allows your Plugin to retrieve large numbers of values in one call.

## **BOGetAttributeNameList**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

OLEVariant array of attribute names.

### **Description**

This method is highly recommended as the best source for determining the attribute names supported by a given business object in any given version of RetailPro.

## **BOGetRepEntityId**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer entity ID.

### **Description**

This returns the entity ID used in the repository to identify the business object definition.

## **BOGetState**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer value of state or an error code as a negative value. Possible state values returned:

Inactive, Browse, Insert, Edit, Opening, or Copy

### **Description**

This returns the State property of the BO specified.

## ***BOGetInstanceActive***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

WordBool

### **Description**

This returns the value of the Active property on the instance dataset.

## ***BOGetActiveDatasetID***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer dataset ID

### **Description**

This returns the value of the repository dataset identifier. This can be used to determine which dataset is the "active" dataset in the business object. BOs have three possible datasets: the list dataset, the instance dataset, and the lookup dataset. As such, the possible values returned are:

- 1 = Invalid BO handle
- 0 = List dataset
- 1 = Instance dataset
- 2 = Lookup dataset

## **BOGetReadOnly**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

True if the business object has been set to read-only.

### **Description**

This returns the State property of the BO specified. Business objects that are read-only cannot be placed in insert or edit mode, their attributes cannot be modified, and calls to Post are rejected.

## **BOSetReadOnly**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

True if the business object has been set to read-only.

### **Description**

This sets the State property of the BO specified to read-only. Business objects that are read-only cannot be placed in insert or edit mode, their attributes cannot be modified, and calls to Post are rejected.

## **BOGetActive**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

True if the business object state is one of Browse, Insert, or Edit.

### **Description**

This returns the Active property of the BO specified and indicates that the business object is ready for data operations.

## BOSetActive

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AValue	WordBool	W	Value to set Active property to.

### Returns

Integer status code. Possible values:

peSuccess, peInvalidBOHandle, peUnableToSetActive

### Description

Setting this to True performs the same operation as BOOpen. Setting it to False performs the same operation as BOClose.

## BOGetUniqueID

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer business object unique identifier, or peInvalidBOHandle

### Description

Every business object instantiated in RetailPro is given a unique, 32-bit, run-time identifier and placed in a business object list. When it's freed, it is removed from that list. Internally in RetailPro, any given business object can be looked up and accessed via its unique identifier. This method is for use by PIAPI developers and is of no real use to Plugin developers.

## BOGetCopyUniqueID

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Copy of integer business object unique identifier, or peInvalidBOHandle

### Description

Every business object instantiated in RetailPro is given a unique, 32-bit, run-time identifier and placed in a business object list. When it's freed, it is removed from that list. Internally in RetailPro, any given business object can be looked up and accessed via its unique identifier. This method is for use by PIAPI developers and is of no real use to Plugin developers.

## **BOGetModified**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

True if any of the business object's attribute values has been modified.

### **Description**

This returns the Modified property of the BO specified and indicates that one or more attribute values has been modified, either by the user via the UI or programmatically.

## **BOGetCommitOnPost**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

WordBool

### **Description**

This returns the current setting of the business object's CommitOnPost flag. Typically set to True by default, this flag controls whether a Commit is done on the dataset immediately following a Post operation. In situations where several rows are inserted, updated, or deleted are performed where all operations should be treated as a single transaction, it is sometimes useful to delay the commit operation until the transaction is completed, with the option to roll back the transaction if any one operation fails.

## **BOSetCommitOnPost**

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

WordBool

**Description**

This sets the business object's CommitOnPost flag. In situations where several rows are inserted, updated, or deleted are performed where all operations should be treated as a single transaction, it is sometimes useful to delay the commit operation until the transaction is completed, with the option to roll back if any one operation fails.

***BOIsAttributeInList*****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)

**Returns**

True if any of attribute is currently contained in the business object's attribute list for displaying a list of records.

**Description**

Not all attributes are available when a business object is initially opened. A frame may clear the attribute list and add only those attributes that are required for display and/or edit purposes. Keep in mind that the end user has the ability, in nearly all lists, to designate the set of columns to display as best suits their business needs. For this reason, it may be necessary to check the list of attributes currently contained in the BO's attribute list before attempting access the attribute.

***BOIsAttributeInInstance*****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)

**Returns**

True if any of attribute is currently contained in the business object's attribute list for displaying a frame for editing a single record.

**Description**

This function is similar to BOIsAttributeInList, but pertains to the attributes used to display a frame with edit fields for managing a single record. Again, the end user has the ability, in nearly all frames, to designate the layout of the frame as best suits their business needs. For this reason, it may be necessary to check the list of attributes currently contained in the BO's attribute list before attempting access the attribute.

## ***BOClearListIncluded***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToClearListIncluded

### **Description**

This clears the list of optional attributes used to construct the list display dataset.

## ***Error! Bookmark not defined.BOClearInstnceIncluded***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToClearInstnceIncluded

### **Description**

This clears the instance used to construct the list display dataset.

## ***BOUpdateListCollections***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

WordBool, True if successful

### **Description**

This causes the business object to update its list of attributes.

## ***BOUpdateInstanceCollections***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
BOHandle	Integer	W	(see common description for BOHandle)



**Returns**

WordBool, True if successful

**Description**

This causes the business object to update its instance collection.

***BOUpdateListDataSet*****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AForceRefresh	WordBool	W	Flag indicating that refresh should be performed regardless of whether attribute list has changed.
AForceDataSetCreation	WordBool	W	Flag indicating that the dataset should be created if it doesn't already exist.

**Returns**

WordBool, True if successful

**Description**

If the list dataset has already been created, this causes the business object to clear its list of attributes and rebuild it, then, if the list of attributes results in a different SQL statement being selected from the repository, the query is resubmitted. If AForceRefresh is set to True, the query is resubmitted regardless of whether the SQL selection was different. If AForceDataSetCreation is set to True, if the dataset does not yet exist, it will be created.

***BOUpdateInstanceDataSet*****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AForceRefresh	WordBool	W	Flag indicating that refresh should be performed regardless of whether the attribute list has changed.
AForceDataSetCreation	WordBool	W	Flag indicating that the dataset should be created if it doesn't already exist.

**Returns**

WordBool, True if successful

**Description**

This operates the same way as BOUpdateListDataSet, but with the business object's instance dataset.

**BOClose****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToClose

**Description**

Given the handle to a business object that is already open, this closes the business object. Subsequent calls to methods that examine the state of the business object such as EOF, BOF, BOGetActive, etc., will reflect that the BO is not open, and calls to retrieve data will return an error code. Internally, this sets Active to False.

**BOOpen****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToOpen

**Description**

Given the handle to a business object that is not already open, this opens the business object. Opening a business object causes it to build a select SQL statement from the repository that best matches the attribute list and data selection criteria, query the database using that SQL statement, and retrieve data from the result set that comes back. Subsequent calls can be made to methods that return data or the state of the business object data. Internally, this sets Active to True.

**BOClear****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToOpen

**Description**

**DO NOT USE.** This method causes the business object to free all its internal structures. Generally speaking, you should never need to use this, and improper use can result in exceptions being returned from the application. In a future version of the PI-API, this method will be reimplemented to clear the values of the attributes currently in the BO's attribute list.

**BOCloseStandAlone****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToClose

**Description**

This method causes the business object to close stand-alone applications.

**BOEdit****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToEdit

**Description**

Puts an open business object that is Active, not Read-Only, and in Browse mode into Edit mode. Calling BOPost will apply changes to the current record.

**BOInsert****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToInsert

**Description**

Puts an open business object that is Active, not Read-Only, and in Browse mode into Insert mode. Calling BOPost will add the attributes values to a new record.

**BODelete****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToDelete

**Description**

For an open business object that is Active, not Read-Only, in Browse mode, and currently positioned on an existing record, this method will attempt to delete the record from the database.

**BOPost****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToPost

**Description**

For an open business object that is in either Insert or Edit mode and shows as Modified, this method attempts to post the data to the database.

**BOPostEx****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
ACancelOnError	WordBool	W	Controls whether an exception is raised or if the operation is simply canceled when a database exception occurs.

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToPostEx

**Description**

This method is identical to BOPost except that, when an error occurs, if the ACancelOnError flag is True, it attempts to perform a database Cancel operation (BOCancel) rather than allowing the exception to surface to the PI-API level. If that flag is False, no Cancel is performed, and the exception surfaces to the PI-API level, which returns a beUnableToPost error code.

**BOCancel****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToCancel

**Description**

On a business object in edit or insert mode, this method attempts to perform a database Cancel operation, discarding any changes and placing the business object in Browse mode.

**BORefresh****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRefresh

**Description**

This method refreshes a business object.

## BOFirst

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToReadFirstRow

### Description

On a business object in Browse mode, this method changes the business object's position to the first row in the active dataset.

## BOLast

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToReadLastRow

### Description

On a business object in Browse mode, this method changes the business object's position to the last row in the active dataset.

## BONext

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToReadNextRow

### Description

On a business object in Browse mode, this method changes the business object's position to the next row in the active dataset. If the dataset is already at the end of the result set (if EOF returns True), this will return peUnableToReadNextRow.

## BOPrior

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToReadPriorRow

### Description

On a business object in Browse mode, this method changes the business object's position to the next row in the active dataset. If the dataset is already at the beginning of the result set (BOF returns True), this will return peUnableToReadPriorRow.

## BOCopy

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToCopy

### Description

On a business object in Browse mode, this method copies the current row into a new row, putting the business object into Insert mode.

## BORollBack

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRollTransactionBack

### Description

On a business object on which changes have been made but Commit hasn't been called, this method rolls back the current transaction. This method is only really useful if the CommitOnPost property on the business object is set to False, allowing for multiple operations in a single transaction.

## BOF

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

True if the business object is positioned at the beginning of the result set.

### Description

This returns True is the business object is Active, Open, its state is Browse, and the active dataset is at the beginning of the result set.

## EOF

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

True if the business object is positioned at the end of the result set.

### Description

This returns True is the business object is Active, Open, its state is Browse, and the active dataset is at the end of the result set.

## BOIsEmpty

### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

True if the business object's result set is empty.

### Description

This returns True is the business object is Active, Open, its state is Browse, and the active dataset is contains no rows. It's the equivalent of BOF and EOF both being True at the same time.



## ***BODisableDataSetControls***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
ASkipIfEdit	WordBool	W	

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRollTransactionBack

### **Description**

Disables data set controls.

## ***BOEnableDataSetControls***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRollTransactionBack

### **Description**

Enables data set controls.

## ***BOUpdateActiveInstance***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AForceRefresh	WordBool	W	Flag indicating that refresh should be performed regardless of whether the attribute list has changed.

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToClearListIncluded

**Description**

This operates similarly to `BOUpdateActiveDataSet`, but on all datasets used by the business object with the exception of the list dataset and any other datasets marked as "standalone."

***BORefreshRecords*****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AForce	WordBool	W	Flag indicating that refresh should be performed regardless of whether the attribute list has changed.
ARefreshCollections	WordBool	W	Flag indicating that refresh of collections should be performed.

**Returns**

WordBool

**Description**

This causes the business object to refresh its list of attributes.

***BOPostAllDataSets*****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
PostCollections	WordBool	W	Flag indicating whether business objects associated with collection attributes should also be refreshed.

**Returns**

Integer status code. Possible values: `peSuccess`, `peInvalidBOHandle`, `peUnableToPostAllDataSets`

**Description**

This method performs a post on all the datasets associated with the specified business object. If `PostCollections` is `True`, the call is propagated to the business objects associated with any collection attributes.

### **BODisableDataSetControls**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

#### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle

#### **Description**

Disables the data-aware controls in the UI and prevents them from updating when you make changes via the PIAPI.

This method can be useful if you have to iterate through a large number of records, such as when working on a document item BO. Calling this method can prevent flicker and improve performance because data does not need to be written to the display.

**Important! MAKE SURE THAT YOU RE-ENABLE THE CONTROLS BEFORE RETURNING TO RPRO.** Put the disable code in a TRY block and put the BOEnableDatasetControls method in the FINALLY block.

Calls to this method can be nested. When all calls to the method are matched to a corresponding call to EnableDatasetControls, the dataset updates data controls and detail datasets.

*Note:* The BOEnable/DisableDatasetControls methods are essentially the same as the Delphi Dataset Enable/DisableControls methods.

### **BOEnableDatasetControls**

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

#### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle

#### **Description**

Enables the data-aware controls in the UI and allows them to update when you make changes via the PIAPI.

After calling BODisableDatasetControls, it is important to call BOEnableDatasetControls before returning to RPRO. If you call BODisableDatasetControls and forget to call BOEnableDatasetControls, data won't update correctly when changes are made in RPRO.

Put the disable code in a TRY block and put the BOEnableDatasetControls method in the FINALLY block.

*Note:* The BOEnable/DisableDatasetControls methods are essentially the same as the Delphi Dataset Enable/DisableControls methods.

## ***BODisableLayoutNotifier***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRollTransactionBack

### **Description**

Disables layout notifier.

RetailPro 9 attributes are notified of changes or events via a method called LayoutChanged. This notification is typically tied to an attribute. If a developer wants to complete a task before this notification occurs, they call BODisableLayoutNotifier. It is crucial that BOEnableLayoutNotifier be called once the task is complete. Not calling BOEnableLayoutNotifier after BODisableLayoutNotifier can leave RPRO in an unexpected state (ie, UI never updates).

Typically these methods are used internally, or in rare occasions by a Plugin which has a specific reason. As always, please use PITest to validate the proper use for your Plugin.

## ***BOEnableLayoutNotifier***

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRollTransactionBack

### **Description**

Enables layout notifier.

RetailPro 9 attributes are notified of changes or events via a method called LayoutChanged. This notification is typically tied to an attribute. If a developer wants to complete a task before this notification occurs, they call BODisableLayoutNotifier. It is crucial that BOEnableLayoutNotifier be called once the task is complete. Not calling BOEnableLayoutNotifier after BODisableLayoutNotifier can leave RPRO in an unexpected state (ie, UI never updates).

Typically these methods are used internally or in rare occasions by a Plugin which has a specific reason. As always, please use PITest to validate there proper use for your Plugin.

## **BOGetPropValById**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AId	Integer	W	One of the following values: 26 Print schema 27 Print schema version 36 Entity caption

### **Returns**

OLEVariant containing the value of the specified property.

### **Description**

This method accesses the properties defined in the repository for the specified business object's entity. The values are typically either string, Boolean, integer, or the ordinal value of an enumerated type.

## **BOGetPosition**

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
BOHandle	Integer	W	(see common description for BOHandle)
APosition	OLEVariant	RW	Bookmark

### **Returns**

APosition as OLEVariant: contains a RetailPro-generated bookmark that can be used with BOLocationPosition to return to the current row.

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToGetPosition

### **Description**

Used on a business object that is open and not empty, this method returns an OLEVariant in the APosition parameter that contains a RetailPro-generated bookmark. The exact contents are unimportant; in database theory, a bookmark is only valid for the current result set. Closing and reopening the business object, and thus the datasets associated with it, invalidates the bookmark. This bookmark can be used to reposition the cursor within the result set back to the row the cursor was on when the bookmark was generated.

## **BOLocatePosition**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
APosition	OLEVariant	W	Bookmark

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToLocatePosition

### **Description**

APosition is a bookmark that was generated using the BOGetPosition method. This bookmark can be used on the same business object without an intervening close and reopen to return dataset cursor to the position it was at when the call to BOGetPosition was made. See description of BOGetPosition.

## **BOSetTempClosingState**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToLocatePosition

### **Description**

This sets the business object's temporary closing state.

## **BOClearTempClosingState**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToLocatePosition

### **Description**

This clears the business object's temporary closing state.

## **BORecalculateAttribute**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRecalculateAttribute

### **Description**

This method tells the business object to recalculate the value for the attribute specified, if it's included in the BO's attribute list.

## **BOClearActiveFakeValues**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRecalculateAttribute

### **Description**

Clears active fake values.

## **BOIsEntity**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AEntityId	Integer	W	Repository entity ID

**Returns**

WordBool, True if matched

**Description**

This method compares the RepEntityID property on the business object with the specified entity ID and returns True if they match.

***BOIsEntityArray*****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AEntityIds	OLEVariant	W	OLEVariant array of repository entity IDs

**Returns**

WordBool, True if one of the IDs matches

**Description**

This method is the similar to BOIsEntity, but scans an OLEVariant array of integer values, returning True if the business object's RepEntityID property matches one of the entity IDs in the array.

***BOIsEntityArray*****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AEntityIds	OLEVariant	W	OLEVariant array of repository entity IDs

**Returns**

WordBool, True if one of the IDs matches

**Description**

This method is the similar to BOIsEntity, but scans an OLEVariant array of integer values, returning True if the business object's RepEntityID property matches one of the entity IDs in the array.



### ***BODisableAccessSecurity***

#### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

#### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRecalculateAttribute

#### **Description**

This disables the business object's access security.

### ***BOEnableAccessSecurity***

#### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

#### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRecalculateAttribute

#### **Description**

This enables the business object's access security.

### ***BODisableSecurity***

#### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

#### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRecalculateAttribute

#### **Description**

This disables the business object's security.

## **BOEnableSecurity**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToRecalculateAttribute

### **Description**

This enables the business object's security.

## **BOFocusAttr**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToFocusAttr

### **Description**

In detail view, this instructs the business object to notify the frame that it needs to put the focus on the first edit control for the specified attribute, assuming the attribute is included in the business object's attribute list and there is an edit control included in the layout for the frame presenting the business object data.

## **BOIncludeAttrForced**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)
AInclude	WordBool	W	

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToFocusAttr

### Description

These include attributes into the business object attribute list. Behind the scenes , the internal methods of **BOIncludeAttrIntoList** and **BOIncludeAttrIntoInstance** are both called and AForced is statically set to True.

### **BOIncludeAttrIntoInstance**

#### Parameters

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)
AInclude	WordBool	W	(see below)
AForce	WordBool	W	(see below)
<i>AInclude</i>	<i>AForce</i>	<i>Description</i>	
False	True	Attribute is <b>removed</b> from <b>ForceAttrIncluded</b> List	
False	False	Attribute is <b>removed</b> from <b>AttrIncluded</b> List	
True	True	Attribute is <b>added</b> to the ForceAttrIncluded List	
True	False	Attribute is <b>added</b> to the <b>AttrIncluded</b> List	

#### Returns

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToFocusAttr

### Description

This adds or removes the attribute into the business object attribute Instance list. The instance List is the list used to determine the attributes that constitutes the Form View in which the attributes are displayed. After BOIncludeAttrIntoInstance is called a **BOUpdateInstanceDataSet** should be called to requery the dataset.

## **BOIncludeAttrIntoList**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)
AInclude	WordBool	W	(see below)
AForce	WordBool	W	(see below)
<i>AInclude</i>	<i>AForce</i>	<i>Description</i>	
False	True	Attribute is <b>removed</b> from <b>ForceAttrIncluded</b> List	
False	False	Attribute is <b>removed</b> from <b>AttrIncluded</b> List	
True	True	Attribute is <b>added</b> to the ForceAttrIncluded List	
True	False	Attribute is <b>added</b> to the <b>AttrIncluded</b> List	

### **Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToFocusAttr

### **Description**

This adds or removes the attribute into the business object attribute list. The List is the list used to determine the attributes that constitutes the List View in which the attributes are displayed. After BOIncludeAttrIntoList is called a **BOUpdateListDataSet** should be called to requery the dataset.

## **BOLocateByAttributes**

### **Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeNames	OLEVariant	W	OLEVariant array of strings
AAttributeValues	OLEVariant	W	OLEVariant array of variants

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToLocateByAttributes

**Description**

This performs a filtered lookup of a record or records based on the names and values specified in the AAttributeNames and AAttributeValues parameters to the TDataset.Locate \* function for scrolling to a specific record. A much faster way to find a record would be to use BOSetFilterAttr.

**BOSortOrderByDomain****Parameters**

<i>Param</i>	<i>Type</i>	<i>Access</i>	<i>Description</i>
BOHandle	Integer	W	(see common description for BOHandle)
ADomain	Integer	W	Attribute Domain ID being filtered.
ASortOrder	Integer	W	
AREopen	WordBool	W	True causes result set to be reloaded.

**Returns**

Integer status code. Possible values: peSuccess, peInvalidBOHandle, peUnableToLocateByAttributes

**Description**

This function performs a sort using the domain ID of the desired attribute. The sort order is an integer value defined in Plugins.tlb that coincides with the ordinal values of the **TSortOrder** type. Setting **AREopen** to True causes the result set to be reloaded.

**BOSortOrderByAttribute****Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)
ASortOrder	Integer	W	
AREopen	WordBool	W	True causes result set to be reloaded.

## Returns

Integer status code. Possible values:

peSuccess  
peInvalidBOHandle  
peUnableToLocateByAttributes

## Description

This function performs the same operation as *BOSortByDomain*, but allows the calling routine to specify an attribute name instead. Internally, this routine uses the attribute name to lookup the domain identifier of the attribute and calls *BOSortByDomain*.

## BOSetFilterAttr

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)
AValue	OLEVariant	W	Attribute value used for filtering.
AOperation	Integer	W	Type of filter being performed.
AFilterData	WordBool	W	True allows filtering; if False no filtering occurs.
AFilterLookup	WordBool	W	Activates the filter.

## Returns

Integer status code. Possible values:

peSuccess  
peInvalidBOHandle  
peUnableToLocateByAttributes

## Description

Allows filtering of a BO's dataset by the specified Attribute Name. To clear filter, call BOSetFilterAttr with AValue unchanged and AOperation set to proNone.

Note: Calling BOSetFilterAttr with a new value for AValue and before clearing the original filter will cause the BOSetFilter to attempt to filter the previously filtered data.

Example:

The Business Object "Customer" Attribute name of "Last Name", BOSetFilterAttr( 0, 'Last Name', 'Smith', proEqual, True, True ) filters the Customer dataset on the 'Last Name' attribute with all values equaling "Smith".)

BOSetFilterAttr( 0, 'Last Name', 'Smith', proNone, True, True ) clears the previous filter.

## BOSetFilter

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
ADomain	Integer	W	Attribute Domain ID being filtered.
AValue	OLEVariant	W	Attribute value used for filtering.
AOperation	Integer	W	Type of filter being performed.
AFilterData	WordBool	W	True allows filtering; if False no filtering occurs.
AFilterLookup	WordBool	W	Activates the filter.

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

Allows filtering of a BO's dataset by the specified Attribute DomainId. To clear filter, call BOSetFilter with AValue unchanged and AOperation set to proNone.

Note: Calling BOSetFilter with a new value for AValue and before clearing the original filter will cause the BOSetFilter to attempt to filter the previously filtered data.

Example:

The Business Object "Customer" Attribute name "Last Name" has a DomainId of 371, using this DomainId, BOSetFilter( 0, 371, 'Smith', proEqual, True, True ) filters the Customer dataset on the 'Last Name' attribute with all values equaling 'Smith'.)

BOSetFilter( 0, 371, 'Smith', proNone, True, True ) clears the previous filter.

## CloneBOHandle

### Parameters

Param	Type	Access	Description
SrcBOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

This adds a reference to a business object wrapper to an adapters list of business object wrappers and returns a new handle.

## Connect

### Parameters

Param	Type	Access	Description
RoleName	PChar	W	Can be left blank
Username	PChar	W	"Pluginuser"
Password	PChar	W	"Plugin"

### Returns

WordBool

### Description

This returns a boolean value indicating whether or not the Plugin session is currently connected to the Retail Pro 9 database. This is only needed if the Plugin is going to access the database directly. It is not an indication of whether or not a business object has access to the database since Plugins use a different session for direct access, which has limited rights granted in order to protect the Retail Pro 9 tables. Before this method can successfully connect to the Retail Pro 9 database, the Plugin must be assigned to a valid BO and that BO must be open using the BOOpen method. The username and password listed above will give the Plugin readonly access to the database.



## ExecSQL

### Parameters

Param	Type	Access	Description
SQL	PChar	W	
Var ResultSet	OLEVariant	W	

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

This executes the specified SQL statement using the previously connected Oracle session. The integer value returned is either -1 for success or an Oracle error code. Please see "Connect" for help on connecting to the Retail Pro database.

## CreateBOByID

### Parameters

Param	Type	Access	Description
BOID	Integer	W	Business object ID.

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

Creates a business object and passes back a handle which the Plugin uses to reference the business object.

## CreateBOByName

### Parameters

Param	Type	Access	Description
BOName	PChar	W	Business object name.

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

Same as CreateChildBO, but accepts a name rather than an integer constant.

## Disconnect

### Parameters

None

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

Disconnects the Plugin database session.

## UpdatePreferences

### Parameters

None

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToLocateByAttributes

### Description

Used to update a limited subset of Retail Pro 9 preference settings. This will initially only access currency rate preferences. DEPRECATED.

## BOGetPref

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
APref	Integer	W	BOPreference value

### Returns

OLEVariant containing the preference value.

### Description

This retrieves the value of a simple preference. These will be a string, integer, or Boolean value. For possible values and descriptions of the returned preference value, see the section in this document titled, "Preference Constants, BOPreference Constants"

## SBSGetPref

### Parameters

Param	Type	Access	Description
ASbs	Integer	W	Subsidiary number
APref	Integer	W	SbsPreference value

### Returns

OLEVariant containing the preference value.

### Description

This retrieves the value of a simple preference associated with the specified subsidiary. These will be a string, integer, or Boolean value. For possible values and descriptions of the returned preference value, see the section in this document titled, "Preference Constants, SbsPreference Constants"

## GetHandleForRootBO

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer handle for root BO or -1 if the specified business object has no root BO.

### Description

In memory, business objects can chain as part of a master detail relationship or as a function of drill down flow. The detail business object keeps a reference to its master business object via the "RootBO" property. When the Plugin requires access to the root BO, this method will register that business object with the Plugin's adapter and pass back a new handle for that business object.

## GetReferenceBOForAttribute

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)

### Returns

Integer handle for reference BO for the specified attribute, or -1 if the attribute has no reference business object or is not a reference attribute.

### Description

In the RetailPro business object hierarchy, detail business objects or lookup business objects are associated to the master business object via reference attributes. These attributes have no values of their own, and serve solely as links to other business objects. A Plugin that requires access to a reference business object can call this method, which will register the business object with the Plugin's adapter, and receive a handle back for that business object.

## GetSecurityPermission

### Parameters

Param	Type	Access	Description
AAApplicationID	Integer	W	The application ID.
APermissionID	Integer	W	The permission ID.

### Returns

WordBool

### Description

GetSecurityPermission uses the application ID to determine which set of permissions to access, then uses the ordinal value of the permission ID cast as the enumerated type stored in the permission set.

## GetAttrPropVal

### Parameters

Param	Type	Access	Description
BOHandle	Integer	W	(see common description for BOHandle)
AAttributeName	PChar	W	(see common description for attribute names)
APropertyID	Integer	W	The property ID.

### Returns

OLEVariant containing the value of the attribute specified.

**Description**

This provides a means of exploring the properties for a specific attribute. This used with BOGetAttributeNameList, this can be used to explore the meta data behind a BO.

Example:

With a parent BO of btCustomer a call to method

BOGetAttrPropVal( 0, PChar( 'Last Name' ), apAttrDomainId ) returns a value of 371.

**GetAttrPropValues****Parameters**

Param	Type	Access	Description
BOHandle	Integer	W	<i>(see common description for BOHandle)</i>
AAttributeName	PChar	W	<i>(see common description for attribute names)</i>
APropertyIDs	OLEVariant	W	The property ID.

**Returns**

OLEVariant containing the value of the attribute specified.

**Description**

This provides the same basic functions as BOGetAttrPropVal but for a list of attributes, making it a bit more efficient when querying large amounts of meta data.

**GetBOEntityProperties****Parameters**

Param	Type	Access	Description
ABOType	Integer	W	The business object type.

**Returns**

XML string

**Description**

GetBOEntityProperties returns an XML string that contains the repository settings for the entity used to describe the business object.

## ***GetBOTypeForEntityId***

### **Parameters**

Param	Type	Access	Description
AEntityId	Integer	W	Repository entity ID.

### **Returns**

Integer entity ID.

### **Description**

This converts an entity ID to a BOType. That BOType can be used to open a child BO. Returns -1 if the entity ID is invalid or is not an entity used by a BO that can be opened as a child BO.

## ***BOCanBeCreated***

### **Parameters**

Param	Type	Access	Description
ABOType	Integer	W	The business object type.

### **Returns**

WordBool

### **Description**

This function returns whether the BO type indicated can be created using the CreateBOByXXX functions. For instance, detail BOs can't be created because they can't function without their master BOs. Instead, you would create the master BO, then obtain a reference to the detail BO that would be generated using GetReferenceBOForAttribute(). Note that if a BO type is invalid, this function simply returns FALSE.

## ***AllBONames***

### **Parameters**

None

### **Returns**

OLEVariant

### **Description**

Returns a list of valid business objects that can be accessed via CreateChildBO and CreateChildBOByName.

## ***ChildBOList***

### **Parameters**

None

### **Returns**

List of integer BO types.

### **Description**

Returns a list of integer BO types of all the child BOs that have been opened or whose handles were cloned or that were obtained as handles for reference attributes by this adapter's Plugin.

## ***Reference***

### **Parameters**

None

### **Returns**

Integer pointer to this object.

### **Description**

Can be used with a little typecasting to debug this object from the Plugin's side of the fence.

## ***Enabled***

### **Parameters**

None

### **Returns**

WordBool

### **Description**

When disabled, the adapter does not communicate signals from Retail Pro to the Plugin and vice versa.

## ***Prepared***

### **Parameters**

None

### **Returns**

WordBool

### **Description**

The following property is currently tied to a data member defined in this class. Later, this property will be moved to the Plugins themselves.

## ***DSCreateVendor***

### **Parameters**

Param	Type	Access	Description
AVendorName	PChar	W	Unique character string key used to identify data belonging to one vendor's Plugin(s). Maximum 20 characters in length.

### **Returns**

PChar SID value, or zero if operation fails.

### **Description**

This creates a vendor entry in the Plugin datastore. The returned SID is used to create dataset definitions and identify all the data used by the identified vendor.

## ***DSModifyVendor***

### **Parameters**

Param	Type	Access	Description
AVendorSid	PChar	W	SID for an existing vendor entry.
ANewName	PChar	W	Unique character string key replacing the current vendor entry name value. Maximum 20 characters in length.

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This replaces the unique vendor key on an existing vendor entry.

## ***DSDropVendor***

### **Parameters**

Param	Type	Access	Description
AVendorSid	PChar	W	SID for an existing vendor entry.

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This deletes a vendor from the vendor table. This call will fail if there is still data in the data store associated with this vendor.



## **DSGetVendorSID**

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
AVendorName	PChar	W	SID for an existing vendor entry.

### **Returns**

Unique string SID associated with existing vendor entry.

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This looks up an existing vendor entry by the vendor name key and returns the vendor's SID.

## **DSCreateDataset**

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetName	PChar	W	Name of the dataset (think table name), unique within the specified vendor.
AVendorSid	PChar	W	SID of an existing vendor entry.
ACMSObject	PChar	W	The name of a table in the RetailPro database that the records for this dataset will be tied to (optional).

### **Returns**

Unique string SID associated with the new dataset.

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This creates a new virtual dataset, associated with an existing vendor entry, and optionally associates records stored in this virtual table with a table the RetailPro (CMS) database.

## ***DSModifyDataset***

Param	Type	Access	Description
ADatasetSid	PChar	W	SID for an existing dataset entry.
ANewName	PChar	W	Unique character string key replacing the current dataset name value. Maximum 20 characters in length. Ignored if blank.
ANewCMSObject	PChar	W	Name of a RetailPro (CMS) table.

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This replaces the unique dataset name key on an existing dataset entry and/or the name of the RetailPro (CMS) table this dataset is associated with.

## ***DSDropDataset***

### **Parameters**

Param	Type	Access	Description
ADatasetSid	PChar	W	SID for an existing dataset entry.

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This deletes a table entry. This call will fail if there is still data associated with the specified dataset.

## ***DSDeleteDatasetData***

### **Parameters**

Param	Type	Access	Description
ADatasetSid	PChar	W	SID for an existing dataset entry.

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

**Description**

This clears out all the data associated with the specified table entry.

***DSInsertIndex*****Parameters**

Param	Type	Access	Description
ADatasetSid	PChar	W	SID for an existing dataset entry.
AREcordSid	PChar	W	SID for an existing data record entry.
ALookupKey1	PChar	W	Lookup value
ALookupKey2	PChar	W	Lookup value (optional=null)
ALookupKey3	PChar	W	Lookup value (optional=null)
ACMSRefKey1	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey2	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey3	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey4	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey5	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.

**Returns**

Unique string SID associated with the new index entry.

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

**Description**

This creates an index for an existing data record. Note that inserting a data record creates an index. The sole purpose for this functionality is to allow creating a single record and associated multiple RetailPro records with it, or providing multiple means for looking up a single data record.

## ***DSModifyIndexLookup***

### **Parameters**

Param	Type	Access	Description
ADataRecordSid	PChar	W	SID for an existing data record entry.
ALookupKey1	PChar	W	Lookup value
ALookupKey2	PChar	W	Lookup value (optional=null)
ALookupKey3	PChar	W	Lookup value (optional=null)

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This modifies the lookup values for all the index entries referencing a single data record.

## ***DSModifyIndexReference***

### **Parameters**

Param	Type	Access	Description
ADataRecordSid	PChar	W	SID for an existing data record entry.
ACMSRefKey1	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey2	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey3	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey4	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey5	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.

### **Returns**

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

## Description

This modifies the values used to look up the data in the associated RetailPro (CMS) table.

## DSDeleteIndex

### Parameters

Param	Type	Access	Description
ADatasetSid	PChar	W	SID for an existing dataset entry.
ARecordSid	PChar	W	SID for an existing data record entry.
ALookupKey1	PChar	W	Lookup value
ALookupKey2	PChar	W	Lookup value (optional=null)
ALookupKey3	PChar	W	Lookup value (optional=null)
ACMSRefKey1	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey2	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey3	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey4	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey5	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.

### Returns

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToDeleteIndex

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### Description

This method deletes a specific index from the indices table. Note that deleting a data record deletes all its associated indices.

## ***DSInsertRecord***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetSid	PChar	W	SID for an existing dataset entry.
ARecordValue	PChar	W	Data to be inserted, up to 4000 bytes.
ALookupKey1	PChar	W	Lookup value
ALookupKey2	PChar	W	Lookup value (optional=null)
ALookupKey3	PChar	W	Lookup value (optional=null)
ACMSRefKey1	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey2	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey3	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey4	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey5	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.

### **Returns**

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToInsertRecord

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This inserts the data record, creates a single index entry, and returns the data record's SID.

## ***DSDeleteRecord***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetSid	PChar	W	SID for an existing data record entry.

### **Description**

This deletes a record from a virtual dataset using the data record SID.

## ***DSUpdateRecord***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ARecordSid	PChar	W	SID for an existing data record entry.
ANewValue	PChar	W	Data to replace record's current value, up to 4000 bytes.

### **Returns**

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToUpdateRecord

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This replaces the data in a data record with the specified value. Note that it is the responsibility of the Plugin to update indices with the correct lookup values. See DSModifyIndexLookup.

## ***DSGetRecordSidByLookup***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetSid	PChar	W	SID for an existing data record entry.
ALookupKey1	PChar	W	Lookup value
ALookupKey2	PChar	W	Lookup value (optional=null)
ALookupKey3	PChar	W	Lookup value (optional=null)

### **Returns**

Integer status code. Possible values:

peSuccess  
 peInvalidBOHandle  
 peUnableToGetRecordSidByLookup

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This returns the first SID for records within a virtual dataset matching the lookup values specified.

## ***DSGetRecordSidByCMSReference***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetSid	PChar	W	SID for an existing dataset entry.
ACMSRefKey1	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey2	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey3	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey4	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey5	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.

### **Returns**



Integer status code. Possible values:

peSuccess  
peInvalidBOHandle  
peUnableToGetRecordSidByCMSReference

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

#### Description

This returns the SID of the first record associated with the index entry that contains the specified CMS reference key values.

### *DSGetRecordBySid*

#### Parameters

Param	Type	Access	Description
ADataRecordSid	PChar	W	SID of data record entry.
ResultSet	OleVariant	R	Variant array (rows) of variant arrays (columns) of variants (values)

#### Returns

Integer status code. Possible values:

peSuccess  
peInvalidBOHandle  
peUnableToGetRecordBySid

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

#### Description

This returns the value of a data record as a result set. (Note: The reason this isn't returned as a simple PChar is to allow for future expansion of this call to accommodate more columns and additional index information.)

## ***DSGetRecordByLookup***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetSid	PChar	W	SID for an existing data record entry.
ALookupKey1	PChar	W	Lookup value
ALookupKey2	PChar	W	Lookup value (optional=null)
ALookupKey3	PChar	W	Lookup value (optional=null)
ResultSet	OleVariant	R	Variant array (rows) of variant arrays (columns) of variants (values)

### **Returns**

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableToGetRecordByLookup

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This returns the first record within a virtual dataset matching the lookup values specified as a variant result set. (Note: The reason this isn't returned as a simple PChar is to allow for future expansion of this call to accommodate more columns and additional index information.)

## ***DSGetRecordByCMSReference***

### **Parameters**

<b>Param</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ADatasetSid	PChar	W	SID for an existing dataset entry.
ACMSRefKey1	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey2	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey3	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey4	Integer	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.
ACMSRefKey5	PChar	W	Value for looking up a record in the associated RetailPro (CMS) table. Optional.

### **Returns**

Integer status code. Possible values:

- peSuccess
- peInvalidBOHandle
- peUnableTo GetRecordSidByCMSReference

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This returns the first record associated with the index entry that contains the specified CMS reference key values as a variant result set. (Note: The reason this isn't returned as a simple PChar is to allow for future expansion of this call to accommodate more columns and additional index information.)

## ***DSGetDatasetSid***

### **Parameters**

Param	Type	Access	Description
AVendorSid	PChar	W	SID for existing vendor entry.
ADatasetName	PChar	W	Name key of existing virtual dataset.

### **Returns**

SID for matching virtual dataset if found, NIL if not found.

Error messages are placed in the adapter's LastErrorCode, LastErrorFunction, and LastErrorMessage properties.

### **Description**

This looks up the named dataset for the specified vendor and, if found, returns the SID.

## ***GetPrimaryID***

### **Parameters**

Param	Type	Access	Description
none	PChar	R	Returns PrimaryID

### **Returns**

Returns the Licensing ClientID. It has the same value on the primary license server (where it matches ClientID) and all secondary license servers. If ClientID is unassigned, an empty string is returned.

### **Description**

When the adapter is available, calling the GetPrimaryID function will return the PrimaryID. If PrimaryID is empty (") update the License in ServiceManager.exe by clicking the "Update License" button. The Update License issue only occurs on Secondary License Servers and is resolved once the Update License button is utilized.

# Plugin Licensing

## ILicense

A new interface called "ILicense" has been added to the API. It has a single property called FeatureIDs and three Methods: Get\_FeatureIDs, Set\_FeatureIDs and Capability,

```
ILicense = interface(IDispatch)
    ['{26579216-8FAD-444D-B39A-FA498477366E}']
    function Get_FeatureIDs: WideString; safecall;
    procedure Set_FeatureIDs(const Value: WideString); safecall;
    function Capability(ACapability: Integer): WordBool; safecall;
    property FeatureIDs: WideString read Get_FeatureIDs write
        Set_FeatureIDs;
end;
```

The "FeatureIDs" property is to be used as a string of comma (",") delimited FeatureIDs.

"Get\_FeatureIDs" and "Set\_FeatureIDs" are the getters and setters of the FeatureIDs property.

The "Capability" function is provided to support future functionality to ILicense without breaking existing Plugins or Plugins which do not have or have not been written with this newer functionality in mind.

In the future, if a new method is added to the ILicense interface, the Capability method would be checked before this new method is called. It would be the responsibility of the Plugin to respond with TRUE if the new functionality is implemented. Capability is not checked with the current implementation of the ILicense interface.

### Implementation:

To utilize this new interface and add licensing to their Plugin, a developer would set its FeatureIDs property with comma delimited list of FeatureId(s). As a service to the Plugin developer, Retail Pro will validate the FeatureId(s) and write only the allocated FeatureId's back to the FeatureIDs property. The Plugin could utilize these allocated FeatureIDs to determine which features are authorized to be used in a multi-featured Plugin (Ex. Plugin "Lite" as opposed to the fully featured Plugin). It is the Plugin developer's responsibility to code access to the allocated features. At Cleanup (i.e. when Retail Pro is terminated), the Plugin Manager would use the FeatureIDs to Deallocate those features.

In the current model, Plugins are deemed "Licensed" even if the FeatureIDs property is empty "" or at least one FeatureID is approved by Retail Pro License Manager. This may not be the case in future releases of Retail Pro.

Here is an example using CustomPluginClasses.

To use the ILicense interface:

1. Add a new CoClass for ILicense.

2. Create a Plugin class for it from CustomPluginClasses.

```
TSampleLicense = class( TCustomLicense )  
Public  
procedure Initialize; override;  
end;
```

3. Override and implement the Initialize method. You only need to implement the Initialize method in the class: { TSampleLicense }

```
procedure TSampleLicense.Initialize;  
begin  
inherited;  
// set the property field with our feature ID's  
fFeatureIDs := cSampleSeatFeatureID + ',' + cSampleSiteFeatureID + ',' +  
cSampleGlobalFeatureID;  
end;
```

4. Add the Plugin to your discover unit like normal and call one of the appropriate support methods as needed in your code:

```
function GetFeatureStatus( FeatureID: String ): LicenseFeatureStatus;  
function IsFeatureAllocated( FeatureID: String ): Boolean;  
function IsAnyFeatureAllocated: Boolean;
```

Please contact Project Services ([ProjectServices@retailpro.com](mailto:ProjectServices@retailpro.com)) for details on obtaining FeatureIDs and adding a Plugin to the Solutions Extension Marketplace.

Implemented in the following Versions:

Version: 9.20.601.072      Date: April 16, 2010

Version: 9.20.504.82      Date: April 16, 2010

## Parent Interfaces

The parent interfaces define methods and properties that are common to a number of descendant interfaces. The parent interfaces are not intended to be used directly when defining a CoClass type. All interfaces in the Plugins type library descend from `IAbstractPlugin`, and so share a common set of methods and properties. This interface does not supply any kind of instantiation, so Plugins should not implement only the `IAbstractPlugin`, and descendant interfaces should not subclass this interface directly.

### **IAbstractPlugin**

#### ***IAbstractPlugin.Adapter***

property Adapter : `IPluginAdapter`;

During discovery, this property is nil. This property is populated by the Plugin manager. The adapter is the Plugin's attachment to the application and enables the Plugin to access business objects, business objects referenced by the business object's attributes, simple preference settings, and a SQL session for accessing the Retail Pro® tablespace directly.

#### ***IAbstractPlugin.Prepare***

function Prepare : `WordBool`; safecall;

This method is called by the Plugin manager once all the Plugins for a given context are instantiated. This function is not called during discovery. The purpose of this method is to allow the Plugin to initialize itself for actual use as opposed to simply identifying itself.

#### ***IAbstractPlugin.Description***

Property Description : `PChar`;

The Plugin is expected to provide a unique and short description for the Plugin, something similar to, "TDG RPro Extensions Invoice Validation Plugin". This is typically only accessed during discovery, although in the case of the `IConfigure` interface, this is the text used on the configuration button in the Retail Pro® preferences editor.

#### ***IAbstractPlugin.GUID***

property GUID : `PChar`;

This is a string representation of the unique value for this Plugin class. The GUID is expected to be unique across the system.

#### ***IAbstractPlugin.Priority***

property Priority : `PluginPriority`

This property allows a Plugin to indicate that it should be loaded before other Plugins. The priority types enumerated in the Plugin type library under "PluginPriority" are `ppLow`, `ppNormal`, and `ppHigh`. Unless the Plugin has a need to be instantiated before any other Plugins in the COM server, this should be set to `ppNormal`.

### ***AbstractPlugin.HandleEvent***

```
function HandleEvent : wordBool; safecall;
```

Unless otherwise noted for a specific Plugin class, the Adapter uses HandleEvent to notify a Plugin that it's time to act. In the case of a validation Plugin, for instance, this method would be called just after an attribute's value changed. The return value controls whether processing continues. If the return value is true (1), processing continues normally. If the return value is false (0), then further event handling, both within the Plugin chain and internally in the application, stops and control is returned to the calling routine. In the case of a validation Plugin, a false return value signifies that the value was rejected.

### ***AbstractPlugin.CleanUp***

```
Procedure CleanUp; safecall;
```

The cleanup method is called at the end of the lifecycle of a Plugin. Most Plugins lifecycles are related to the Business Objects in which they are associated (btInvoice, btSalesOrderBO...). But this is not always the case; IServerPlugin, ILoginPlugin have different lifecycles.

To ensure that the cleanup method works correctly for your Plugin, test its behavior using PITest.dll. PITest.dll is a Plugin that implements most of the Plugin API. It is available on the developers FTP site. There is a corresponding tool call PITestCongig.exe used to configure PITest. It is also on the FTP site.

Remember that it is the responsibility of the Plugin to:

- Clear all structures
- Deallocate memory
- Nill the adapter

*Reference:* See the "Plugin Lifecycle" section of this document.

## **IBOPlugin**

### ***IBOPlugin.BusinessObjectType***

```
property BusinessObjectType : Integer;
```

This property is expected to return one of the enumerated values of type BusinessObjectType from the Plugins type library.



## IAAttributePlugin

### IAAttributePlugin.AttributeName

property AttributeName : PChar;

This string value identifies the name of the attribute within the business object that the Plugin is connected to.

### IAAttributePlugin.AttrPermissionEnabled

function AttrPermissionEnabled(APermission, APermType: Integer): Integer;

This function allows for an Attributes permissions to be temporarily overridden. While the Plugin is active and set for a specific AttributeName and focus is on this particular attribute, the permissions can be overridden. This allows an attribute to be set to READONLY or MASKED (like a password).

**APermission** : Defined in Plugins\_TLB.pas file see AttrPermission.

**APermType** : Defined in Plugins\_TLB.pas file see PermissionType.

There are three valid return integer values: (see AttributePermissions)  
apeEnabled(0), apeDisabled(1), apeIgnore(2).

**Example1:** To make a simple attribute such as the Customer "Last Name"  
READONLY:

First in the "Initialize" procedure set FAttributeName := 'Last Name'; and FBOType := btCustomer. Then add the following function.

```
function TAttributeAssignment.AttrPermissionEnabled(APermission,
APermType: Integer): Integer;
begin
  { Simple Attribute Example }
  result := apeIgnore;
  if ( APermission = papEdit ) and
    (( APermType = aptNormal )      or ( APermType = aptOverride ))
  then
    result := apeDisabled;
end;
```

The following code snippets demonstrate the labeled UI functionality.

```
{ MASKED but editable }
{ if ( APermission = papAccess ) and
  (( APermType = aptNormal ) or ( APermType = aptOverride )) then
  result := apeDisabled;
}
{ READONLY and MASKED }
{ if (( APermission = papAccess ) or ( APermission = papEdit )) and
  (( APermType = aptNormal ) or ( APermType = aptOverride )) then
  result := apeDisabled;
```

**Example2:** To make single cell within a Collection attribute READONLY, in this case we will make the AdjmentsMemo.Comments collection attribute "Comment" READONLY when the attribute "Comment #" = 10.

First in the "Initialize" procedure set FAttributeName := 'Comment'; and FBOType := btAdjMemoCommentBO. Then add the following function.

```
function TAttributeAssignment.AttrPermissionEnabled(APermission,
APermType: Integer): Integer;
var
    Id      : integer;
Begin
{ Collection Attribute Example }
    result := apeIgnore;
    Id := FAdapter.BOGetAttributeValueByName( 0, PChar( 'Comment #' ));
    if ( Id = 10 ) and ( APermission = papEdit ) and
        (( APermType = aptNormal )      or ( APermType = aptOverride ))
    then
        result := apeDisabled;
end;
```

NOTE: A word of caution, RetailPro calls AttrPermissionEnabled for every Permission (5) and PermissionType (4) for each AttributeAssignment Plugin upon focus being set on that attribute. For simple attribributes this is not an issue but for large collections you may notice a lag when scrolling through each row.

## IHardwarePlugin

### *IHardwarePlugin.CashRegister*

#### In Delphi

```
property CashRegister : IPOSStationAdapter;
```

#### Notes

This is populated with a reference to the POS station object. This object acts as the adapter and event sink for any and all point of sale hardware devices.

## Base Interfaces

When the Plugin manager is performing the Plugin context discovery process, it checks to see if the Plugins implement specific interfaces. Only classes that implement these interfaces are recognized and instantiated by the Plugin manager. When creating a Plugin specific type library, the Plugin writer will need to subclass these interfaces.

These Plugin interfaces are recognized as being instantiated in connection with the creation of a specific business object, as indicated by the value of the Plugin's `BusinessObjectType` method.

### IAttributeValidationPlugin

Inherits from **IAttributePlugin**. This Plugin is triggered just before an edit is applied to the attribute it is connected to.

#### **IAttributeValidationPlugin.HandleEvent**

Function `HandleEvent` : `WordBool`; `safecall`;

In all Plugins, returning `True` from `HandleEvent` indicates that processing should stop, and in the `IAttributeValidationPlugin`'s case, it also causes RetailPro to set focus back to the attribute (if visible) and abort further processing.

#### **IAttributeValidation.PluginCapability**

function `PluginCapability`(`ACapability`: `Integer`): `wordBool`; `safecall`;

Please see `PluginCapability` definition on page 87.

### IAttributeAssignmentPlugin

Inherits from **IAttributePlugin**. This Plugin is triggered when the attribute it is connected to has been modified.

### IEntityUpdatePlugin

#### **IEntityUpdate.BeforeUpdate**

Function `BeforeUpdate` : `WordBool`; `safecall`;

This method is called just prior to committing new or modified values to the Retail Pro database for a given business object. The return value indicates whether to continue with the data operation. A return value of `true` (1) allows the operation to continue. A return value of `false` (0) aborts the data operation. If a value of `false` is returned, in a UI operation the user is presented with the data as entered. It is suggested that a dialog indicating the error be presented to the user.

### **IEntityUpdate.AfterUpdate**

Procedure AfterUpdate; safecall;

This method is called just after data has been committed to the Retail Pro database for a given business object.

### **IEntityUpdate.OnCancel**

procedure OnCancel; safecall;

This method is called just before data entered into the current Entity is destroyed.

### **IEntityUpdate.BeforeInsert**

function BeforeInsert: wordBool; safecall;

This method is called just before a new Entity is created. At this point the entity has not been created and a query of an attribute will contain data from the Entity previously created in which the Retail Pro database was currently pointed. A return value of true (1) allows the operation to continue. A return value of false (0) aborts the data operation. If a value of false is returned, in a UI operation the user is presented with the data as entered. It is suggested that a dialog indicating the error be presented to the user.

### **IEntityUpdate.AfterInsert**

procedure AfterInsert; safecall;

This method is called just after a new Entity is created. At this point the entity has been created and a query of an attribute will contain the initialized data for the new Entity.

### **IEntityUpdate.BeforeEdit**

function BeforeEdit: wordBool; safecall;

This method is called just before the Entity mode is set to edit. A return value of true (1) allows the operation to continue. A return value of false (0) aborts the data operation. If a value of false is returned, in a UI operation the user is presented with the data as entered. It is suggested that a dialog indicating the error be presented to the user.

### **IEntityUpdate.AfterEdit**

procedure AfterEdit; safecall;

This method is called just after the Entity mode is set to edit.

### **IEntityUpdate.BeforeDelete**

function BeforeDelete: wordBool; safecall;

This method is called just before an Entity is deleted. A return value of true (1) allows the operation to continue. A return value of false (0) aborts the data operation. It is suggested that a dialog indicating the error be presented to the user.

### **IEntityUpdate.AfterDelete**

procedure AfterDelete; safecall;

This method is called just after an Entity is deleted.

### **IEntityUpdate.BeforeReversal**

*Note:* The BeforeReversal and AfterReversal methods are limited in support to the Business Objects that can be reversed. (**Invoices, Adjustment Memos, PI Sheets, Slips, SO Deposits, Transfer Orders, Vouchers** )”

```
function BeforeReversal(AEmpId: Integer; var AComment1, AComment2: PChar): wordBool;
```

This method is called just before an Entity is Reversed. A return value of true (1) allows the operation to continue. A return value of false (0) aborts the data operation. It is suggested that a dialog indicating the error be presented to the user.

### **IEntityUpdate.AfterReversal**

*Note:* The BeforeReversal and AfterReversal methods are limited in support to the Business Objects that can be reversed. (**Invoices, Adjustment Memos, PI Sheets, Slips, SO Deposits, Transfer Orders, Vouchers** )”

```
procedure AfterReversal(ANewDocSid: PChar);
```

This method is called just after an Entity is Reversed. The parameter ANewDocSid is the SID in which the reversal was recorded.

### **IEntityUpdate.BeforeCopy**

```
function BeforeCopy: WordBool; safecall;
```

This method is called just before an Entity is Copied. A return value of true (1) allows the operation to continue. A return value of false (0) aborts the data operation. It is suggested that a dialog indicating the error be presented to the user.

### **IEntityUpdate.AfterCopy**

```
procedure AfterCopy; safecall;
```

This method is called just after an Entity is Copied.

### **IEntityUpdate.PluginCapability**

```
function PluginCapability(ACapability: Integer): wordBool; safecall;
```

Please see the IPluginAdapter > PluginCapability definition.

## ItemAddRemovePlugin

*Note:* This interface does not use the `HandleEvent` method, but instead implements event handlers called `BeforeEmptyItemAdd`, `AfterEmptyItemAdd`, `BeforeItemRemove`, `AfterItemRemove`, `BeforeChooseProcessItem`, `AfterChooseProcessItem`, `BeforeNewItemCommit`, `AfterNewItemCommit`.

### **IItemAddRemovePlugin.AfterChooseProcessItem**

```
procedure AfterChooseProcessItem(ItemBOHandle : Integer; ADocSid : PChar;  
AItemData : PChar ) : wordBool; safecall;
```

This event handler is called when the user is adding an item using item lookup in the items grid, using the SO/PO/TO Items dialog, or when exiting a Choose/Edit function. In other words, when an item is added to the document, this event is called, no matter how the item was selected. `ItemBOHandle` contains the handle for the detail business object. The document SID is passed in `ADocSID`. Because the BO is not used to store the selected items, the data RetailPro is about to process into the database is passed as an XML string in the `AItemData` property. This method will generally be used to react to the successful insertion/update of the detail. See `BeforeChooseProcessItem` for more details.

### **IItemAddRemovePlugin.AfterItemAdd**

```
function AfterItemAdd(ItemBOHandle : Integer) : wordBool; safecall;
```

Deprecated. See `AfterEmptyItemAdd`, which replaces this method.

### **IItemAddRemovePlugin.AfterEmptyItemAdd**

```
function AfterEmptyItemAdd(ItemBOHandle : Integer) : wordBool; safecall;
```

This event handler is called just after an empty detail entry is added to the detail business object. A handle for that detail business object is passed in for use with the `BOxxxx` methods defined in this interface's ancestor, `IBOPlugin`. This event handler will generally be used to default values into the detail's attributes.

### **IItemAddRemovePlugin.AfterItemRemove**

```
function AfterItemRemove(ItemBOHandle : Integer) : wordBool; safecall;
```

This event handler is called just after a detail has been deleted from the database. At this point in time, the detail has already been eliminated.

### **IItemAddRemovePlugin.BeforeChooseProcessItem**

```
function BeforeChooseProcessItem(ItemBOHandle : Integer; ADocSid : PChar; var  
AItemData : PChar; var AModified : WordBool) : WordBool; safecall;
```

This event handler is called when the user is adding an item using item lookup in the items grid, using the SO/PO/TO Items dialog, or when exiting a Choose/Edit function. In other words, when an item is added to the document this event is called, no matter how the item was selected. ItemBOHandle contains the handle for the detail business object. The document SID is passed in ADocSID. Because the BO is not used to store the selected items, the data RetailPro is about to process in the database is passed as an XML string in the AItemData property. If the item is a new item, the ItemPos element in the ItemData XML will be -999. If it's an existing item and the user has changed the quantity, the ItemPos will be some value other than -999. At this point in time, the Plugin can alter the values in the XML and pass them back. If it does so, AModified should be set to True (1) before returning. Passing back False (0) as the value of AModified signifies that the XML was not altered and any changes to the XML will be ignored. If this method passes back False (0) as its return value, the insertion of the item is rejected and the function continues with the remaining selected items. This method will generally be used to validate items being processed and, if appropriate, alter them as needed.

### **IItemAddRemovePlugin.BeforeItemAdd**

```
function BeforeItemAdd(ItemBOHandle : Integer) : WordBool; safecall;
```

Deprecated. See BeforeEmptyItemAdd, which replaces this method.

### **IItemAddRemovePlugin.BeforeEmptyItemAdd**

```
function BeforeEmptyItemAdd(ItemBOHandle : Integer) : WordBool; safecall;
```

This event handler is called just prior to creating an empty detail entry in detail business object. At this point in time, the business object is not attempting to post the detail to the database. This simply creates an empty slot. A handle for that detail business object is passed in for use with the BOxxxx methods defined in this interface's ancestor, IBOPlugin. The return value indicates whether to continue with the operation. A return value of true (1) allows the operation to continue, and return value of false (0) aborts the operation. This event handler will generally be used to control whether it's appropriate to add a detail to the document.

### **IItemAddRemovePlugin.BeforeNewItemCommit**

```
function BeforeNewItemCommit(ItemBOHandle : Integer) : WordBool; safecall;
```

This event handler is called just prior to posting a detail in detail business object to the database. A handle for that detail business object is passed in for use with the BOxxxx methods defined in this interface's ancestor, IBOPlugin. The return value indicates whether to continue with the operation. A return value of true (1) allows the operation to continue, and return value of false (0) aborts the operation. This event handler will generally be used to validate the row for consistency and completeness.

**IItemAddRemovePlugin.BeforeItemRemove**

```
function BeforeItemRemove(ItemBOHandle : Integer) : WordBool; safecall;
```

This event handler is called just prior to removing a detail from the detail business object and the database. A handle for that detail business object is passed in for use with the BOxxxx methods defined in this interface's ancestor, IBOPlugin. The return value indicates whether to continue with the operation. A return value of true (1) allows the operation to continue, and return value of false (0) aborts the operation. This event handler will generally be used to validate the deletion of the row and reject it if appropriate to do so.

**IItemAddRemovePlugin.PluginCapability**

```
function PluginCapability(ACapability: Integer): WordBool; safecall;
```

Please see the IPluginAdapter > PluginCapability definition.

**IItemAddRemovePlugin.BeginChooseProcessItems**

Because a user can choose multiple items from the choose edit button which will be returned to the document, BeginChooseProcessItems and EndChooseProcessItems were added to signal the API developer that this event has occurred.

```
function BeginChooseProcessItems(AContext: OleVariant): WordBool; safecall;
```

This method is Called immediately after user returns from Choose/Edit Items frame. AContext is RESERVED for future use and is intended to contain a variant array of values that may be useful in making decisions before the item list is processed. Currently will contain Null. The return value indicates whether to continue with the data operation. A return value of true (1) allows the operation to continue, and return value of false (0) aborts the data operation. In the case returning false, in a UI operation the user is presented with the data as entered. It is suggested that a dialog indicating the error be presented to the user.

**IItemAddRemovePlugin.EndChooseProcessItems**

```
procedure EndChooseProcessItems(AResults: OleVariant); safecall;
```

Called immediately after the item list has been processed after leaving the Choose/Edit Items frame. AResults, is RESERVED for future use and is intended to contain a variant array of values that indicate the final state after the item list is processed. Currently will contain Null.



## ISideButtonPlugin

### ISideButtonPlugin.Caption

property Caption : PChar;

Defines the text caption that appears on the button.

### ISideButtonPlugin.PictureFileName

property PictureFileName : PChar;

This string is the name of the Windows bitmap picture (.bmp) that is to be displayed as the glyph on the button. The name is assumed to be a filename located in the \RetailPro9\Plugins subdirectory. Bitmaps for side buttons should be 36x36 pixels in size and use a 24-bit color palette.

### ISideButtonPlugin.ButtonEnabled

property Enabled : WordBool;

This property defines whether a button is enabled. This property is checked frequently by Retail Pro, and is assumed to be a dynamic value. The Plugin can change the value as appropriate, controlling access to the Plugin function behind the button.

### ISideButtonPlugin.UseLayoutManager

property UseLayoutManager : WordBool;

This property controls whether or not the button can be placed using Layout Manager. If this value is true (1), the user has the .option of placing the button on the menu using Layout Manager. For now, this value should always be true (1).

### ISideButtonPlugin.Checked

property Checked : WordBool;

This property controls whether the button has a check mark on it. This property is checked frequently by Retail Pro and is assumed to be dynamic.

### ISideButtonPlugin.Hint

property Hint : PChar;

This property controls the "fly-over" hint that appears when the user places the mouse cursor over the side button. This value is only checked when the button control is instantiated and placed in the side menu. It is not dynamic in this version.

### ISideButtonPlugin.LayoutActionName

property LayoutActionName : PChar;

This string value defines a name for the control that is unique across all Plugins and across all controls used internally in Retail Pro. It is recommended that the Plugin use a name constructed of the server name, the Plugin name, and if needed, a numeric value.

### ISideButtonPlugin.PluginCapability

function PluginCapability(ACapability: Integer): WordBool; safecall;

Please see the IPluginAdapter > PluginCapability definition.

## ICustomAttributePlugin

Inherits from IAttributePlugin. This interface is used to create a calculated or non-persistent attribute on a form for a given business object. Retrieval, persistence, and any necessary calculation of the value in the attribute is entirely up to the Plugin.

## IPrintPlugin

Inherits from IBOPlugin. This Plugin is notified whenever a document is printed. Internally in Retail Pro, documents are first exported to XML and then passed to the Retail Pro print engine. This Plugin is given access to this XML object before it's processed by the print engine. It is up to the Plugin to understand the structure of the XML object and alter, insert, or remove text as needed within that object.

### IPrintPlugin.XMLDoc

property XMLDoc: IXMLDOMDocument;

When the HandleEvent method is called for the IPrintPlugin Plugin, this property will have been supplied with an interface reference to an XML DOM document contained within Retail Pro. The Plugin can make any necessary changes to that document prior to its submission to the Retail Pro print engine.

### IPrintPlugin.PluginCapability

function PluginCapability(ACapability: Integer): wordBool; safecall;

Please see the IPluginAdapter > PluginCapability definition.

## ITenderPlugin

Inherits from IBOPlugin. This Plugin is triggered at various times during the collection and processing of tender information for invoices and sales orders. Note that the tender Plugin interface was architected with electronic fund transfer (EFT) validation in mind. The tender process for an invoice is treated as a transaction within which a number of tenders can be used. The assumption is that the Plugin will keep a list of the tenders for a transaction in local memory, and that those tenders can be accessed by Retail Pro via a tender ID that is returned by the AddTender method. The ITenderPlugin methods are designed to add tender entries to that list, remove tenders if needed, settle the tenders with a third party solution, and either commit the transaction or cancel it and roll back the tenders.

*Note:* If the Plugin implementing this interface doesn't perform EFT validation, it still must implement these methods and return valid responses.

### **ITenderPlugin.AddTender**

```
function AddTender(TenderType : Integer; var Data : PChar) : Integer; safecall;
```

This event is called when a tender is added to the tender grid in Retail Pro. The tender type indicates the type of tender (credit card, gift card, debit card, etc.). The Data parameter contains the text representation of a name value pair string list constructed by Retail Pro containing all the pertinent information for the tender in question. It is assumed that the Plugin will not alter the contents of the string buffer pointed to by the Data parameter. Instead, the contents should be copied to a buffer local to the Plugin. Upon return, a pointer to that buffer should be put into the Data parameter. Retail Pro copies the result value out of the Plugin buffer into a buffer local to Retail Pro. Careful respect of these rules will prevent buffer overruns and memory corruption.

In Delphi, the following is how the Data parameter should be handled by a Plugin:

```
function TMyTenderPlugin.AddTender(TenderType:Integer; var Data:PChar):
Integer;
var
    LocalBuffer : String;
begin
    // strpas() creates an ANSI string from a PChar variable
    LocalBuffer := strpas(Data);
    {...}
    // The LocalBuffer string can then be read, modified, etc.
    {...}
    Data := pchar(LocalBuffer);
end;
```

Upon return, the adapter for the **ITenderPlugin** will copy the string value pointed to by Data into a buffer local to Retail Pro. Note that the return value is the tender ID that is used with any subsequent methods in the Plugin that require a tender ID. Any error code is expected to be contained in the string pointed to by the Data parameter upon return.

### **ITenderPlugin.CancelTransaction**

```
function CancelTransaction : Integer; safecall;
```

This event is called when a user cancels out of an invoice, and notifies the Plugin to cancel a transaction and all the tenders that were associated with it.

### **ITenderPlugin.CommitTransaction**

```
function CommitTransaction : Integer; safecall;
```

This event tells the Plugin that the user committed the transaction and that it can now clear its buffers, close any open handles, and reinitialize.

### **ITenderPlugin.IsTenderActiveInBatch**

```
function IsTenderActiveInBatch(TenderID : PChar) : wordBool; safecall;
```

This function returns true (1) if the tender indicated by the tender ID is still in the tender batch.

### **ITenderPlugin.RemoveTender**

```
function RemoveTender(TenderIndex : Integer) : Integer; safecall;
```

This function removes, and rolls back, if necessary, a tender added earlier by AddTender. The TenderIndex is the return value from that AddTender call.

**ITenderPlugin.StartTransaction**

```
function StartTransaction : wordBool; safecall;
```

This function allows the Plugin to initialize, allocate, and open anything it needs during the processing of a single transaction. Returns true (1) if initialization occurred without error.

**ITenderPlugin.PluginCapability**

```
function PluginCapability(ACapability: Integer): wordBool; safecall;
```

Please see the IPluginAdapter > PluginCapability definition.

## IConfigure

The IConfigure interface implements functionality that allows the plug-in to present configuration dialogs needed for any plug-in specific configuration.

### Methods

**procedure TEFTComObject.ConfigureAll;**

This method is called when the user double clicks the plug-in name in the workstation preferences screen for configuring plug-ins. This method should be self contained presenting the configuration screens and returning control when the configuration is complete.

**procedure ConfigurePlugin(var PluginGUID: TGUID);**

This method is presented for completeness and is not implemented in EFT plug-ins.

## IEFTPlugin

Inherits from IBOPlugin. Used for authorization of electronic fund transfers, including credit card, debit card, check, gift card, etc.

**IEFTPlugin.Settle**

```
function Settle(TenderType: Integer; All: wordBool): wordBool; safecall;
```

Called to trigger batch settlement.

**IEFTPlugin.StartTransaction**

```
function StartTransaction: wordBool; safecall;
```

Begins a transaction.

**IEFTPlugin.AddTender**

```
function AddTender(TenderType: Integer; var Data: PChar): Integer; safecall;
```

Adds a tender to a transaction and processes it.

**IEFTPlugin.CommitTransaction**

```
function CommitTransaction: wordBool; safecall;
```

Finishes a successful transaction.

### **IEFTPlugin.CancelTransaction**

function CancelTransaction: WordBool; safecall;

Finishes a failed transaction. Successful tenders should be voided.

### **IEFTPlugin.RemoveTender**

function RemoveTender(TenderIndex: Integer): WordBool; safecall;

VOIDS a successful tender and removes it from the transaction.

### **IEFTPlugin.IsTenderActiveInBatch**

function IsTenderActiveInBatch(TenderID: PChar): WordBool; safecall;

Checks to see if a given tender is still in the open batch awaiting settlement.

### **IEFTPlugin.IsOnline**

function IsOnline(TenderType: Integer): WordBool; safecall;

Determines if the gateway is reachable from the Plugin.

### **IEFTPlugin.PluginCapability**

function PluginCapability(ACapability: Integer): WordBool; safecall;

## **ITenderDialoguePlugin**

Inherits from IBOPlugin. This Plugin defines logic for replacing one of Retail Pro's internal tender logins. It tells RetailPro which tender type it intends to replace, and when the user clicks the button for that tender type from within a POS document, the Plugin is called in place of RetailPro's default dialogue. Note that, unlike most other Plugin types, these cannot be chained; once an ITenderDialoguePlugin is called, the tender dialogue event is considered handled, further processing stops, and control is returned to the POS document entry screen.

### **Persistence**

Retail Pro does not store every field for every tender type. The following describes which fields are stored in the database for a given tender type.

<i>Tender Type</i>	<i>Fields</i>
Cash	Taken Given Amount Currency_ID Manual_Remark
Check	Amount Currency_ID Doc_No Auth Chk_Type Transaction_ID Manual_Remark

<i>Tender Type</i>	<i>Fields</i>
Foreign Checks	Amount Currency_ID Doc_No Auth Manual_Remark
Credit Cards	Amount Currency_ID Doc_No Auth Crd_Exp_Month Crd_Exp_Year Transaction_ID Crd_Type Crd_Present AVS_Code L2_Result_Code Signature_Map Manual_Remark
COD	Amount Currency_ID Manual_Remark
Charge	Amount Currency_ID Charge_Net_Days Charge_Disc_Days Charge_Disc_Perc Manual_Remark
Store Credit	Amount Currency_ID Doc_No Auth Transaction_ID Check_Type Manual_Remark
SO Deposit	Amount Currency_ID Transaction_ID Chk_Type Manual_Remark
Payments	Amount Currency_ID Pmt_Date Pmt_Remark Manual_Remark
Gift Certificates	Amount Currency_ID Doc_No Auth Manual_Remark

<i>Tender Type</i>	<i>Fields</i>
Gift Cards	Amount Currency_ID Doc_No Auth Transaction_ID Crd_Present Gft_Crd_Balance Manual_Remark
Debit	Amount Currency_ID Doc_No Auth Crd_Exp_Month Crd_Exp_Year Transaction_ID Crd_Present Manual_Remark
Travelers Checks	Amount Currency_ID Doc_No Auth Transaction_ID Chk_Type Manual_Remark

### Business Rules for Tender Display and Storage

When using the ITenderDialoguePlugin, the Plugin must respect the business rules for tenders in RetailPro. The dollar amounts in the tender's "give" and "take" columns, as displayed in the tenders grid, are always positive -- there's no artificial switching of the signs -- so the tally of the amounts equals the amount of the document.

The tender subtotal ("amt") field is not necessary for RPro processing or reporting purposes. The subtotal can be derived from the given and taken columns.

While, historically, the given and taken columns were used to express the handling of foreign currency only, X/Z Out reporting requires the current definition of those columns, which is the giving and taking of all tenders. The subtotal is, on the other hand, needed for ECM and 8 series compatibility.

The way in which the amount of a given tender is applied is determined solely by the document type. Receipt is positive, Return is negative.

The following shows the way in which the amounts are displayed and stored:

<i>Doc Type</i>	<i>Use</i>	<i>Doc Amt</i>	<i>Taken</i>	<i>Given</i>	<i>Tender Amt</i>
Receipt	UI Display	80	100	20	80
	DB Storage	80	100	20	80

Return	UI Display	80	0	80	80
	DB Storage	80	80	0	80

### Suppressing the DeleteEFTTender Dialog

When a user chooses to delete a Tender the "Do you want to delete?" dialog is shown. If you have an ITenderDialoguePlugin associated with this Tender, the DeleteTender event will be fired before the tender is actually deleted. In some cases, based on your definition the tender behavior, a deletion may not be possible. Plugin developers can suppress this dialog by returning FALSE in response to the DeleteTender event. Internally an exception is thrown to halt the deletion of that tender.

One exception to this rule is if the ITenderDialoguePlugin handles the PluginCapability "tdcUseDefaultDialog". In this, the DeleteTender event will fire but the return value is ignored. The thinking here is if the Plugin is using the default retail pro dialog then it shouldn't be able to suppress the delete dialog.

#### ITenderDialoguePlugin.Clear

```
procedure Clear; safecall;
```

Clear tells the Plugin to initialize all the property values (except for the tender type) to zero, empty string, or a default value, prior to creating a new tender entry.

#### ITenderDialoguePlugin.HandleEvent

```
function HandleEvent : wordBool; safecall;
```

HandleEvent is used to tell the Plugin to execute. During the HandleEvent call, it is expected, although not required, that the Plugin present a dialogue, collect tender information from the user, obtain EFT authorization if necessary, and return all the data pertinent to the tender type back to RetailPro. A return value of True tells RetailPro that the tender operation completed successfully and the tender entry is to be saved in RetailPro's database. A return value of false indicates that the tender operation was cancelled.

#### ITenderDialoguePlugin.TenderType

```
property TenderType: Integer read Get_TenderType;
```

This read-only property is read during discovery and indicates which RetailPro tender dialogue is being replaced.

#### ITenderDialoguePlugin.Amount

```
property Amount: Currency read Get_Amount write Set_Amount;
```

This is the amount to be tendered. This typically represents the outstanding, untendered balance on the POS document. The Plugin can change this amount during tender processing.

#### ITenderDialoguePlugin.Taken

```
property Taken: Currency read Get_Taken write Set_Taken;
```

This is the amount taken from the customer. If the untendered balance is \$15, the customer gives the clerk a \$20 bill and receives \$5 in change, this property would be \$20. The taken minus the given should be equal the tendered amount.



### **ITenderDialoguePlugin.Given**

property Given: Currency read Get\_Given write Set\_Given;

This is the amount taken from the customer. If the untendered balance is \$15, the customer gives the clerk a \$20 bill and receives \$5 in change, this property would be \$5. The taken minus the given should be equal the tendered amount.

### **ITenderDialoguePlugin.Authorization**

property Authorization: PChar read Get\_Authorization write Set\_Authorization;

When a tender requires EFT authorization or, at a minimum, manual authorization, this property should contain the authorization code. Note that at the time of this writing, the authorization code is limited to 26 chars.

### **ITenderDialoguePlugin.TransactionID**

property TransactionID: PChar read Get\_TransactionID write Set\_TransactionID;

TransactionID records the identifier for an EFT authorization. Different gateways and processors have varying names for this field, e.g. troutd, transid, token, etc., but the purpose is to identify an authorization. Limited to 26 chars.

### **ITenderDialoguePlugin.AVSCode**

property AVSCode: PChar read Get\_AVSCode write Set\_AVSCode;

In conjunction with an EFT authorization, this is the AVS code.

### **ITenderDialoguePlugin.L2ResultCode**

property L2ResultCode: PChar read Get\_L2ResultCode write Set\_L2ResultCode;

In conjunction with an EFT authorization, this is the L2 result code.

### **ITenderDialoguePlugin.AccountNumber**

property AccountNumber: PChar read Get\_AccountNumber write Set\_AccountNumber;

This is the account or identifying number of the tender medium. On a magnetic stripe card, this is the card number, on a check, it would be the check number, etc. Limited to 30 characters.

### **ITenderDialoguePlugin.FirstName**

property FirstName: PChar read Get\_FirstName write Set\_FirstName;

The first name of the individual whose name appears on the tender medium.

### **ITenderDialoguePlugin.LastName**

property LastName: PChar read Get\_LastName write Set\_LastName;

The last name of the individual whose name appears on the tender medium.

### **ITenderDialoguePlugin.WorkPhone**

property WorkPhone: PChar read Get\_WorkPhone write Set\_WorkPhone;

The work phone of the individual presenting on the tender medium. Limited to 11 characters.

### **ITenderDialoguePlugin.HomePhone**

property HomePhone: PChar read Get\_HomePhone write Set\_HomePhone;

The home phone of the individual presenting the tender medium. Limited to 11 characters.

### **ITenderDialoguePlugin.StateProvince**

property StateProvince: PChar read Get\_StateProvince write Set\_StateProvince;

The state or province of the individual presenting the tender medium. Limited to 2 characters.

### **ITenderDialoguePlugin.PostalCode**

property PostalCode: PChar read Get\_PostalCode write Set\_PostalCode;

The ZIP code or postal code of the individual presenting the tender medium. Limited to 10 characters.

### **ITenderDialoguePlugin.DriversLicense**

property DriversLicense: PChar read Get\_DriversLicense write Set\_DriversLicense;

The driver's license number of the individual presenting the tender medium. Limited to 30 characters.

### **ITenderDialoguePlugin.DriversLicenseExpDate**

property DriversLicenseExpDate: PChar read Get\_DriversLicenseExpDate write Set\_DriversLicenseExpDate;

The expiration date of the driver's license of the individual presenting the tender medium in YYYYMM or YYYYMMDD format. (If DD is left off, "01" is assumed.)

### **ITenderDialoguePlugin.DateOfBirth**

property DateOfBirth: PChar read Get\_DateOfBirth write Set\_DateOfBirth;

The date of birth of the individual presenting the tender medium, in YYYYMMDD format.

### **ITenderDialoguePlugin.CheckCompany**

property CheckCompany: PChar read Get\_CheckCompany write Set\_CheckCompany;

The name of the company on a business check.

### **ITenderDialoguePlugin.CheckType**

property CheckType: Integer read Get\_CheckType write Set\_CheckType;

The type of the check. 0 = personal check, 1 = business check.

### **ITenderDialoguePlugin.CardType**

property CardType: Integer read Get\_CardType write Set\_CardType;

This integer value indicates which card type is being used. This equates to the card types

### **ITenderDialoguePlugin.CardName**

property CardName: PChar read Get\_CardName write Set\_CardName;

The name of the issuing card company. RetailPro uses the CREDIT\_CARD table to associate this name with a credit card type, and this value is one of the values in the CRD\_NAME column. This value is not stored in the database.

### **ITenderDialoguePlugin.CardExpDate**

property CardExpDate: PChar read Get\_CardExpDate write Set\_CardExpDate;

The date the credit card expires, in YYYYMM format.

### **ITenderDialoguePlugin.CardNormalSale**

property CardNormalSale: WordBool read Get\_CardNormalSale write Set\_CardNormalSale;

This indicates that the card was used for a sale rather than a return, balance check, etc.

### **ITenderDialoguePlugin.CardPresent**

property CardPresent: WordBool read Get\_CardPresent write Set\_CardPresent;

This property indicates whether the card was present for the transaction, as opposed to a call-in or on-line purchase.

### **ITenderDialoguePlugin.CardProcessingFee**

property CardProcessingFee: Currency read Get\_CardProcessingFee write Set\_CardProcessingFee;

This property is set to any additional card processing fee that was assessed during the EFT authorization.

### **ITenderDialoguePlugin.CardSignatureMap**

property CardSignatureMap: PChar read Get\_CardSignatureMap write Set\_CardSignatureMap;

This property is set to a signature point array as defined in the OPOS specification. The string returned should be a wide string. Every two wide characters represent 16 bit X/Y coordinates on a grid, with \$FFFF representing a pen lift. These coordinates can be used to plot the signature to a canvas prior to printing or conversion to a different format. This is limited to 2000 wide chars in length.

### **ITenderDialoguePlugin.GiftCardTraceNo**

property GiftCardTraceNo: PChar read Get\_GiftCardTraceNo write Set\_GiftCardTraceNo;

This property is set to the trace number for a gift card.

### **ITenderDialoguePlugin.GiftCardInternalReference**

property GiftCardInternalReference: PChar read Get\_GiftCardInternalReference write Set\_GiftCardInternalReference;

This property is set to the internal reference number for a gift card.

### **ITenderDialoguePlugin.GiftCardBalance**

property GiftCardBalance: Currency read Get\_GiftCardBalance write Set\_GiftCardBalance;

This property indicates the remaining gift card balance following authorization of the transaction.

### **ITenderDialoguePlugin.ChargeNetDays**

property ChargeNetDays: Integer read Get\_ChargeNetDays write Set\_ChargeNetDays;

This property contains the "net days" information of a charge transaction. A charge might be 30/15/10, meaning that the total balance must be paid off in 30 days, but if the charge is paid in 15 days, the customer will be given a 10% discount.

### **ITenderDialoguePlugin.ChargeDiscountDays**

property ChargeDiscountDays: Integer read Get\_ChargeDiscountDays write Set\_ChargeDiscountDays;

This property contains the "discount days" information of a charge transaction. See "ITenderDialoguePlugin.ChargeDiscountDays" for an example.

### **ITenderDialoguePlugin.ChargeDiscountPercent**

property ChargeDiscountPercent: Single read Get\_ChargeDiscountPercent write Set\_ChargeDiscountPercent;

This property contains the "discount percent" information of a charge transaction. See "ITenderDialoguePlugin.ChargeDiscountDays" for an example.

### **ITenderDialoguePlugin.PaymentDate**

property PaymentDate: PChar read Get\_PaymentDate write Set\_PaymentDate;

This is the date a payment is made.

### **ITenderDialoguePlugin.PaymentRemark**

property PaymentRemark: PChar read Get\_PaymentRemark write Set\_PaymentRemark;

This is the optional remark describing a payment. For any tender type that isn't payment, this is also used as the comment describing the tender. This value is stored in the invoice tender data in the database using the "manual\_remark" field and is displayed as the overriding comment in the tender details list. Maximum length is 40 characters.

### **ITenderDialoguePlugin.CurrencyType**

property CurrencyType: Integer read Get\_CurrencyType write Set\_CurrencyType;

This is the currency type. This equates to the "currency\_id" field in the "currency" table in the database.

### **ITenderDialoguePlugin.CVV2Result**

property CVV2Result: PChar read Get\_CVV2Result write Set\_CVV2Result;

The CVV2 result pertains to the EFT authorization procedure, indicating a match or mismatch between the CVV2 value input by the clerk and the one associated with the card in the processor's database.

### **ITenderDialoguePlugin.EFTForced**

property EFTForced: WordBool read Get\_EFTForced write Set\_EFTForced;

This flag indicates whether an EFT authorization was manually entered rather than going through a processor. This may be the case when the connection to the gateway is down or unavailable.

### **ITenderDialoguePlugin.Remark**

property Remark: PChar read Get\_Remark write Set\_Remark;

The Remark property, if populated, overrides RetailPro's comment construction logic. This can be used to supply descriptive text to be displayed in RetailPro's tender grid and tender details dialogue. All tender types can make use of this property. The maximum length is 40 characters.

### **ITenderDialoguePlugin.EncryptAccountNumber**

This flag has been DEPRECATED.

### **ITenderDialoguePlugin.PluginCapability**

function PluginCapability(ACapability: Integer): wordBool; safecall;

Please see the IPluginAdapter > PluginCapability definition.

## **CustomPluginClass**

The CustomPluginClass is a set of classes that implement the Plugin.tlb interfaces so that Plugin developers do not have to do it themselves.

Although it is not a requirement that a Retail Pro 9Plugin implement these classes, by inheriting from one of the custom classes, your Plugin receives these benefits:

- Simplifies the development of RetailPro Plugins.
- You only need to implement (override) the particular methods relevant to your Plugin and its problem domain. (ex: "HandleEvent" and "Initialize" for TCustomSidebuttonPlugin)
- Facilitates logging. Logging has been added to the TCustomAbstractPlugin class. All CustomPluginClasses descend from TCustomAbstractPlugin. To enable logging, set the "LogLevel" property to something greater than logNone and in each method in which logging is desired, add a call to the overloaded method "Log". The log is written to the local workstation\Log path. Also, debug logging has been implemented in each parent class method. By setting the LogLevel to logDEBUG and running your new Plugin (assuming, it compiles, and is in the "Plugins" directory along with it's Manifest file, etc.) a minimally implemented Plugin will log all calls to that Plugin so you can see what is being sent to the Plugin and in what order.giving you an idea of what data and what order each method is sending to the Plugin.
- CustomPluginClasses is supported by RetailPro. Any changes to the API will be reflected in CustomPluginClasses. All you would need to do is save the CustomPluginClasses unit into your shared directory (over writing the existing version) and then next time you compile your Plugin the new API feature(s) will be included.
- CustomPluginClasses is supported in Delphi and .NET. The .NET implementation is written in C#. The beauty of .NET is that other .NET languages can inherit from assemblies developed in other .NET languages. So VisualBasic.net can utilize the CustomPluginClasses.dll assembly as well and there is no need to create a version of CustomPluginClasses in VisualBasic.

### ***Delphi implementation:***

#### **uPluginDiscover.pas**

Place "uPluginDiscover" in a shared library path. There is no need to make any changes to this unit. (see uPluginGlobals)

#### **uPluginGlobals.pas**

Use "c\_PluginModuleVersion" to set your Plugin version, You can change the default messages of "c\_ClassFactoryExceptionMsg" and "c\_ClassFactoryExceptionCls" to format Class Factory error message to your liking.

The array "c\_ClassFactoryArray" - used by the Discover unit, but declared here so you never have to modify the discover unit first add the name of the constant created by the TLB wizard for the CoClass you added to your project TLB, with a @ before the constant name then add the name of the class in which you implemented that CoClass add them in pairs for each CoClass you declared in your project TLB. NOTE: Don't forget to increment the upper array size. The lower array boundary MUST be left at ZERO!

If when creating the tlb file for your project you use the naming conventions provided here for the Discover class (Plugin\_Discover), you will never have to change the discover line of the array (first line) as the TLB constant will be called CLASS\_Plugin\_Discover.

## **.NET Plugins – C# Example**

### **CSInvoiceSideButton**

Some key points in getting your Plugin up and running:

In References:

1) Add:

- a) CustomPluginClasses
- b) MSXML
- c) RetailPro.Plugins

In Using clause:

2) Add

- a) RetailPro.CustomPluginClasses;
- b) System.Runtime.InteropServices;

In Properties:

3) Application Tab:

OutPut type = Class Library  
Click "Assembly Information" and check "Make assembly COM-Visible"

4) Build Tab:

In the Output section, check "Register for COM interop".

5) Signing Tab:

Check "Sign the assembly"  
In the combobox titled "Choose a strong name key file:"  
Select <New...> (Add a name and password. !!!NOTE: Make sure to right it down!!!)

Plugin Manifest file:

6) In C# the "ServerName" that you will place in the manifest file should be the same as the Namespace where the Server resides. In this case it should look like this "CSInvoiceSideButton.Discover". This can be confusing when you have a "Project name", "Assembly name", "Default name". If you put the wrong server name in the manifest file, Retail Pro will complain. To be sure what RetailPro is looking for, you can search the Registry for "YourServerName". When you find the "YourServerName.DiscoveryClassName", that is the name to go into the Manifest file. Also, Retail Pro does not support "AutoReg" flag with .Net com servers, you have to register them manually or create a script. This is note pertains to step #4 above, having the IDE register your Plugin for you. Please see the Microsoft help files for guidance if you are going to register it yourself with the Gacutil and RegAsm utilities.

It is very important to decorate your Plugin classes with the following Attributes:

```
[ClassInterface(ClassInterfaceType.None)]  
[Guid("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")]
```

The GUID MUST be generated for YOUR Plugin! Do not copy these Guid's as they will conflict with your next Plugin you create. Each Plugin must have its own unique GUID. To generate a guid for your Plugin, choose the "Tools" > "Create GUID" menu, GUID format #4, copy and paste in two places: In your static constant class as above and in the GUID attribute as below. Make sure to do it for each and every Plugin class you implement.

### Debugging a C# Plugin:

To debug a .Net Plugin while Retail Pro is running, follow these steps:

On the "My Project" -> "Debug" tab, click the "Start Action" radio button labeled "Start external program" and enter the path of your Retail Pro exe ("C:\RetailPro9\RPRO9.exe").

In "Start Options" you can put in any Retail Pro command line parameters you want and you can leave the working directory empty.

In "Enable Debuggers" check "Enable unmanaged code debugging" Add any break points you want in your code, press (F5) to start debugging.

You can also follow these steps to Attach to a process: Start Retail Pro. Once it is running, in the VS IDE, navigate to "Debug">"Attach to Process..."> menu and select these options:

- Transport: "Default" Attach to: "
- Managed code" Available Processes: "
- RPRO9.exe" Check -> "
- Show Process in all sessions"

Now add the Sidebutton to the Invoice Menu. If you set a breakpoint in the HandleEvent method, then click the "C# SideButton" you just added in Retail Pro, your code will stop at the breakpoint previously set.

Initialize method:

Make sure to call the base.Initialize(), there are some important initializations that happen there.

```
public override void Initialize()
{
    base.Initialize();
    // Added additional initialization code here...
}
```



## Visual Basic Plugin example:

### VBInvoiceSideButton

#### Note on VB Namespaces:

Do not create a namespace here, set your Namespace in "MyProject" -> "Application" tab "Root Namespace". While it is perfectly legal to create "Local" Namespace here, your COM server will be Registered in Windows as "RootNamespace.LocalNamespace.Discover" RetailPro will complain because it is looking for the format of "ServerName.DiscoverClassName"

This note pertains to step #4 below, having the IDE register your Plugin for you. Please see the Microsoft help files for guidance if you are going to register it yourself with the Gacutil and RegAsm utilities.

Some key points in getting your Plugin up and running:

#### In References:

##### 1) Add:

- a) CustomPluginClasses
- b) MSXML
- c) RetailPro.Plugins

#### In Imports clause:

##### 2) Add

- a) RetailPro.CustomPluginClasses;
- b) System.Runtime.InteropServices;

#### In My Project:

##### 3) Application Tab:

OutPut type = Class Library  
Click "Assembly Information" and check "Make assembly COM-Visible"

##### 4) Build Tab:

In the Output section, check "Register for COM interop".

##### 5) Signing Tab:

Check "Sign the assembly", in the combobox titled "Choose a strong name key file:" select <New...> (Add a name and password. **!!!NOTE: Make sure to right it down!!!**)

In Plugin Manifest file:

6) ServerName:

In VB, the "ServerName" that you will place in the manifest file should be the same as the "Root Namespace" Applications Tab. In this case it should look like this "VBInvoiceSideButton.Discover". This can be confusing when you have a "Project name", "Assembly name", "Local Namespace". If you put the wrong server name in the manifest file, Retail Pro will complain. To be sure what RetailPro is looking for, you can search the Registry for "YourServerName". When you find the "YourServerName.DiscoveryClassName", that is the name to go into the Manifest file. Also, Retail Pro does not support "AutoReg" flag with .Net com servers, you have to register them manually or create a script.

It is very important to decorate your Plugin classes with the following Attributes:

```
<GuidAttribute(Discover.CLASS_DiscoverPlugin)>
```

The GUID MUST be generated for YOUR Plugin! Do not copy these GUIDs as they will conflict with your next Plugin you create. Each Plugin must have its own unique GUID. To generate a GUID for your Plugin, choose the "Tools" > "Create GUID" menu, GUID format #4, copy and paste in two places: In your Public constant and in the GUID attribute as below. Make sure to do it for each and every Plugin class you implement.

### **Debugging a VB Plugin:**

To debug a .net Plugin while Retail Pro is running, follow these steps:

On the "My Project" -> "Debug" tab, click the "Start Action" radio button labeled "Start external program" and enter the path of your Retail Pro exe ("C:\RetailPro9\RPRO9.exe").

In "Start Options" you can put in any Retail Pro common line parameters you want and you can leave the working directory empty.

In "Enable Debuggers" check "Enable unmanaged code debugging". Add any break points you want in your code, press (F5) to start debugging.

Initialize method:

Make sure to call the base.Initialize(), there are some important initializations that occur.

```
Public overrides Sub Initialize()  
    MyBase.Initialize();  
    // Added additional initialization code here...  
End Sub
```

## Enumerated Values

The enumerated values in Plugins.tlb fall into two categories:

**BusinessObjectTypes:** The BusinessObjectType constants tell Retail Pro what business object a user-interface Plugin should be attached to.

**Error codes:** Passed back in lieu of a business object handle in calls to the adapter that would normally return a positive integer business object handle.

### Business Objects

Constant Name	Description
btGeneric	Used internally in Retail Pro; do not use this value.
btAdjustmentMemo	Adjustment document.
btAdjustmentMemoItem	Adjustment document details.
btASNVoucher	Voucher.
btAlloc	Item allocations to stores.
btCustomer	Customer information.
btDepartment	Department information.
btFrmrVoucher	Former Voucher.
btItem	Item detail on receipt.
btInvoice	Invoice (receipt).
btMemo	Memo document.
btMovNote	Inventory movement notification document (formerly "SubLocs").
btMovOrder	Inventory movement order document (formerly "SubLocDefs").
btPendVoucher	Pending Voucher.
btPOHist	Purchase Order history.
btPOTerms	Purchase Order terms.
btSlip	Transfer slip.
btSOHist	Sales Order history.
btSOTerms	Sales Order details.
btPhysicalInventory	Physical Inventory.
btTOItemAllocation	Transfer Order Item Allocation.
btVendor	Vendor.
btAllocBO	

Constant Name	Description
btTender	Point of Sale Tender.
btSalesOrderBO	Sales Order document.
btSOItemBO	Sales Order item information.
btVoucher	Voucher document.
btVoucherItemBO	Voucher details
btPOSDocBO	A generic point of sale document (receipts and sales orders).
btPOSDocItemBO	A generic item on a point of sale document (receipts and sales orders).

### Error Codes

The following are the error codes in Plugins.tlb. Note that some of the values are negative. With the exception of peSuccess, these are passed back in lieu of a business object handle in calls to the adapter that would normally return a positive integer business object handle.

Constant Name	Value	Description
peSuccess	-1	Operation completed with no errors.
peUnableToCreateBO	-2	Unable to create business object.
peUnsupportedBO	-3	The business object type specified is not supported. The business object type must be one of the BusinessObjectType constants.
peInvalidBOHandle	-4	The handle passed in was not valid for this Plugin.
peGenericFailure	0	Any non-specific failure.
peUnableToUpdateDataSetRecords	1	Call to BOUpdateDataSetRecords failed.
peUnableToSetAttributeValue	2	Call to BOSetAttributeValue failed. That attribute may not be intrinsically modifiable or you do not have write access to it.
peUnableToSetAttributeValues	3	Call to BOSetAttributeValues failed. See description above.
peUnableToClearListIncluded	4	Call to BOClearListIncluded failed.
peUnableToClearInstancedIncluded	5	Call to BOClearInstanceIncluded failed.

Constant Name	Value	Description
peUnableToUpdateListCollections	6	Call to BOUpdateListCollections failed.
peUnableToUpdateInstanceCollections	7	Call to BOUpdateInstanceCollections failed.
peUnableToOpen	8	Call to BOOpen failed.
peUnableToClose	9	Call to BOClose failed. Either the business object wasn't open at that time or the business object could not be closed without first saving/canceling modifications.
peUnableToCloseStandalone	10	Call to BOCloseStandalone failed.
peUnableToEdit	11	Call to BOEdit failed.
peUnableToInsert	12	Call to BOInsert failed.
peUnableToDelete	13	Call to BODelete failed. The business object may not have been open, or the application may not allow deletions to the current record.
peUnableToPost	14	Call to BOPost failed. The business object may have failed data validations, or it may not have been in edit/insert mode.
peUnableToPostExecute	15	Call to BOPostEx failed. See description above.
peUnableToCancel	16	Call to BOCancel failed.
peUnableToRefresh	17	Call to BORefresh failed.
peUnableToReadFirstRow	18	Call to ReadFirstRow failed.
peUnableToReadNextRow	19	Call to ReadNextRow failed.
peUnableToReadPriorRow	20	Call to ReadPriorRow failed.
peUnableToCopyRow	21	Call to BOCopyRow failed
peUnableToRollTransactionBack	22	Call to BORollTransactionBack failed. The business object might not have had a transaction in progress at that time, or does not allow transactional rollbacks at that time.
peUnableToDisableDataSetControls	23	Call to BODisableDataSetControls failed.

Constant Name	Value	Description
peUnableToEnableDataSetControls	24	Call to BOEnableDataSetControls failed
peUnableToUpdateActiveInstance	25	Call to BOUpdateActiveInstance failed.
peUnableToRefreshRecord	26	Call to BOREfreshRecord failed. The application may not allow this operation because a transaction is in progress or the business object is in edit/insert mode.
peUnableToPostAllDataSets	27	Call to BOPostAllDataSets failed. Some validation may have failed or the business rules don't allow a post operation at this time.
peUnableToDisableLayoutNotifier	28	Call to BODisableLayoutNotifier failed.
peUnableToEnableLayoutNotifier	29	Call to BOEnableLayoutNotifier failed.
peUnableToGetPosition	30	Call to BOGetPosition failed.
peUnableToSetTempClosingState	31	Call to BOSetTempClosingState failed.
peUnableToClearTempClosingState	32	Call to BOClearTempClosingState failed.
peUnableToRecalculateAttribute	33	Call to RecalculateAttribute failed. If the attribute requires that data be read from the database to recalculate, transactional issues may prevent this operation.
peUnableToClearActiveFakeValues	34	Call to BOClearActiveFakeValues failed.
peUnableToDisableAccessSecurity	35	Call to BODisableAccessSecurity failed.
peUnableToEnableAccessSecurity	36	Call to BOEnableAccessSecurity failed.
peUnableToDisableSecurity	37	Call to BODisableSecurity failed.
peUnableToEnableSecurity	38	Call to BOEnableSecurity failed.
peSessionNotConnected	39	Some operation was attempted that needed to be performed with an active session. You must successfully call Connect first.

Constant Name	Value	Description
peSuccessNoResults	40	The SQL statement processed without reporting error codes, but did not return a result set.
peAttributeNotFound	41	The Attribute supplied to the method was not found in that BO Type.
peUnableToReadLastRow	42	Unable to read last row.
peUnsupportedFunction	43	Unsupported function.
peUnableToLocateByAttributes	44	Call to BOLocateByAttributes failed.

## Preference Constants

The following are constants used internally in Retail Pro 9 to access simple preference values, but are not included in the type library due to the fact that this list will change from time to time.

**BO Preference Constants:** These constants are used with the BOGetPref function to return an OLEVariant representation of the preference value. The BOGetPref function only returns simple preferences. Complex preferences or preferences based on SQL tables are not returned. See BOGetPref for more details on the function of the method. Please refer to the document "RPro 9.1 Preference Constants" for descriptions of values returned for each of the preference constants.

**Subsidiary Preference Constants:** The SbsPreference constants are used with the SbsGetPref function to return an OLEVariant representation of the preference value. The SbsGetPref function only returns simple preferences. Complex preferences or preferences based on SQL tables are not returned. See SbsGetPref for more details on the function of the method. The following list should not be considered current or comprehensive. Updates will be made available as possible via addenda to this document. Constant names are listed solely for documentation purposes. When accessing the properties listed, use the "Value" in the call to BOGetPref.

### BOPreference Constants

Constant Name	Value	Return Value Datatype	Description
bpCustFullName	0	Boolean	True = require full customer first and last names
bpUseVat	1	Boolean	True = enable VAT calculations

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpVatOpts	2	Bitmask	Bits: 1 = Use ICM 2 = Round extended up 3 = Tax code sub calc
bpEncryptSchema	3	Boolean	
bpDefaultCommCod	4	Integer	Invoice comm. Code
bpInvenPriceMethod	5	Integer	Values: 1 = None 2 = Rounding 3 = Adjusting
bpInvenPriceMethodAffect Manual	6	Boolean	Use inventory price method
bpAccessInactive	7	Boolean	Access inactive records
bpGMAffectCost	8	Boolean	True = margins affect cost
bpTaxedMargin	9	Boolean	True =
bpTaxedMarkup	10	Boolean	True =
bpTaxedCoeff	11	Boolean	True =
bpDefaultStore	12	Boolean	True =
bpAllowDuplicateALU	13	Boolean	True = allow duplicate inventory ALU codes
bpUseNCharsForStyleSid	14	Integer	Length of style
bpNotConsolidateDocItems	15	Boolean	True =
bpEnableSerialNoTrack	16	Boolean	True =
bpDefaultSerialNoTrackLevel	17		
bpRecordVendNameToDescr2	18		
bpItemDocLookupItemNo	19	Boolean	True = include item number in inventory lookup filter
bpItemDocLookupUPC	20	Boolean	True = include UPC in inventory lookup filter



<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpItemDocLookupALU	21	Boolean	True = include ALU in inventory lookup filter
bpItemLookupMode	22	Integer	Bits: 1 = None 2 = Item number 3 = UPC 4 = ALU
bpNegValueFormat	23	Pchar	Formatting mask for displaying negative values.
bpUseThousandSeparatorForQuantity	24	Boolean	True = quantities displayed using thousands separator
bpUseThousandSeparatorForCurrency	25	Boolean	True = currency amounts displayed using thousands separator
bpUseQtyDecimals	26	Boolean	True = quantities displayed with 2 decimal places
bpUseCurrencySymbol	27	Boolean	True = currency amounts displayed with currency symbol.
bpPriceDecimals	28	Integer	0-5, number of decimal places to display in price
bpCostDecimals	29	Integer	0-5, number of decimal places to display in cost
bpTaxAmtDecimals	30	Integer	0-5, number of decimal places to display in tax
bpASNChooseEditItems	31		
bpASNMakeReturn	32		
bpItemsNegativeQty	33	Boolean	True = prevents user from inputting negative qty
bpAddrLookupByZIP	34	Boolean	True = perform city/state lookup when entry made in ZIP field
bpZipLookupAddyMask	35	Pchar	Character mask used to parse address data entered into postal code field.

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpStyleSidGenSource	36	Integer-Integer Bit Mask	Bits: 1 = System 2 = UPC 3 = ALU
bpCharsInStyleSidGen	37	Byte	Number of characters in the SID
bpNextASNSeqNo	38	PChar	The next available ASN sequence number
bpNextPOSeqNo	39		The next available purchase order number
bpDefaultStoreDocShip	40	PChar	On purchase orders, the default "ship to" store number
bpDefaultStoreDocBill	41	PChar	On purchase orders, the default "bill to" store number
bpCurrentUserEmplId	42	PChar	Employee ID of currently signed on user.
bpPO_Inst1	43	PChar	Purchase order shipping and handling instructions
bpPO_Inst2	44	PChar	Purchase order shipping and handling instructions
bpPO_Inst3	45	PChar	Purchase order shipping and handling instructions
bpPO_Inst4	46	PChar	Purchase order shipping and handling instructions
bpPO_Inst5	47	PChar	Purchase order shipping and handling instructions
bpDuplicateCustId	48	Boolean	True = allow duplicate customer IDs
bpDuplicateCustName	49	Boolean	True = allow duplicate customer names
bpNextCustIDSeqNo	50	PChar	The next available customer ID sequence number
bpMultiVendorDoc	51	Boolean	True = documents may contain multiple vendors

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpWarningWhenNewCost GreaterThanOld	52	Boolean	True = when validating inventory cost field, display a warning when the new cost for an item is higher than the previous value
bpNewCostWarningThresh oldPerc	53	Integer	Percent threshold before cost warning is displayed
bpDefaultStoreDoc	54	PChar	Default store number to place on documents
bpDontUseLastCost	55	Boolean	True = no default value for item cost
bpOrdrCaseOnly	56	Boolean	True = only allow ordering by the case
bpCaseQtyRoundedMetho d	57	Integer	When determining the number of cases based on a quantity of units  0 = round the number of cases up to the next whole integer  1 = round the number of cases down  2 = normal rounding to the nearest integer
bpAllwEdtCustID	58	Boolean	True = allow the user to modify the customer ID
bpUseTradeDiscount	59	Boolean	True = use trade discount
bpAdjMemoOnQty	60	Boolean	True = allow user to access quantity field on adjustment memo
bpAdjMemoOnPrice	61	Boolean	True = allow user to access price field on adjustment memo
bpAdjMemoOnCost	62	Boolean	True = allow user to access cost field on adjustment memo
bpAdjQtyStored	63	Boolean	True = save store quantities on the adjustment

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpCommentRequired	64	Boolean	True = requirement comments on documents
bpPOPrefix	65		
bpCheckItemSerialNumber	66	Boolean	True = validate serial numbers on documents
bpNextAdjMemoSeqNo	67	PChar	Next adjustment memo sequence number
bpUpdateOrderCost	68	Boolean	True = update the order cost on a voucher when posting changes
bpUpdatePrice	69	Boolean	True = update price on voucher when posting changes
bpCostMethod	70	Integer	Values: 1 = not defined 2 = Average 3 = Overwrite 4 = Leave
bpNextTOSeqNo	71	PChar	Next transfer order sequence number
bpLedgerItemInfo	72	PChar	Default item info on a ledger entry
bpUserName	73	PChar	User name of currently logged in user
bpDataBaseName	74	PChar	Database name the application is accessing
bpUserId	75	PChar	User ID of the currently logged in user.
bpUserEmplId	76	PChar	Employee ID of the currently logged in user
bpUseTax2	77	Boolean	True = multiple tax areas are used and some of those tax areas use different codes for the same type of merchandise

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpUsePWTFForTax2	78	Boolean	True = 1st price will be included when calculating a 2nd tax on price
bpNextReturnVouSeqNo	79	PChar	Next return voucher sequence number
bpNextReceiveVouSeqNo	80	PChar	Next receive voucher sequence number
bpNextStoreCreditSeqNo	81	PChar	Next store credit sequence number
bpNextSlipSeqNo	82	PChar	Next slip sequence number
bpUseSeparateVouSeq	83	Boolean	True = Generate a separate voucher sequence number when copying a voucher
bpReqVouRefToPO	84	Boolean	True = require a purchase order reference on vouchers
bpTwoStepApproval	65	Boolean	True = require user to review invoice prior to approving it.
bpUseVendorInvoice	86	Boolean	True = allow updates to invoice vendor
bpAutoRetrieveLedger	87	Boolean	True = In the ledger tree view, enables retrieval of ledger information when the tree view changes
bpCheckAvailQty	88	Integer	Values: 1 = check available 2 = check on hand 3 = do not check
bpAllowPOPastCancelDate	89	Boolean	True = allow a purchase order to be processed after cancel date has expired
bpAllowUpdateDoc	90	Boolean	True = allow updates to existing documents
bpAllowReverseDoc	91	Boolean	True = allow documents to be reversed

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpAutoGenSlipSetup	92	Integer	Values: 1 = Send on-hand 2 = Send ordered 3 = Optimize
bpUseAutoUPC	93	Boolean	True = automatically assign next UPC from sequence
bpUseAutoALU	94	Boolean	True = automatically assign next ALU from sequence
bpNextAutoUPC	95	PChar	Next UPC in sequence
bpNextAutoALU	96	PChar	Next ALU in sequence
bbpAutoALUMask	97	Boolean	Mask for generating ALU numbers in sequence
bpAllowPackageItemsInDoc	98	Boolean	True = allow package items (kits) onto invoice
bpAllowDuplicateItemsInDoc	99	Boolean	True = allow an inventory item to be used in more than one invoice detail
bpPOMask	100	PChar	Purchase order sequence number mask
bpAutoPONoGen	101	Boolean	True = autogenerate purchase order numbers using the PO mask
bpUseMultiTax	102	Boolean	True = calculate tax using multiple breaks
bpPreventNegativeCost	103	Boolean	True = prevents user from entering negative cost
bpOverwriteCustTitle	104	Boolean	True = allow users to make free-form entries in the Title field of customer records
bpOverwriteVendTitle	105	Boolean	True = allow users to make free-form entries in the Title field of vendor records
bpNextSOSeqNo	106	PChar	Next sales order sequence number

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpNextReturnInvSeqNo	107	PChar	Next return invoice sequence number
bpNextInvSeqNo	108	PChar	Next invoice sequence number
bpUseSeparateInvSeq	109	Boolean	True = use separate numbers for each invoice
bpPIType	110	Integer	Values: 0 = Simple 1 = Multizone 2 = Multizone and multistore
bpPIInitBy	111	Integer	Values: 0 = UPC 1 = Item number 2 = ALU
bpPIThresholdCost	112	Integer	
bpPIUseFilters	113	Boolean	True = Use filters when displaying list of physical inventory
bpPIStripZeroes	114	Boolean	
bpPIAutoUpdateVariance	115	Integer	
bpUseSmartScan	116	Integer	
bpSmartscanGroup1	117	PChar	
bpSmartscanGroup2	118	PChar	
bpSmartscanGroup3	119	PChar	
bpMultiVendorPO	120	Boolean	Flag controlling multi-vendor purchase order number generation
bpAllwSprdGlobDisc	121	Boolean	True = allow discount spread
bpPOSFlagDefMnu1	122	Integer	Default value for UDF field #1 on POS document
bpPOSFlagDefMnu2	123	Integer	Default value for UDF field #2 on POS document

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpPOSFlagDefMnu3	124	Integer	Default value for UDF field #3 on POS document
bpRndTenderTypes	125	Integer bit mask	Bits: 1 = None 2 = Cash 3 = COD 4 = Deposit 5 = Store credit 6 = Foreign currency 7 = Check 8 = Credit card 9 = Charge 10 = Payments 11 = Gift certificate 12 = Gift card 13 = Debit card 14 = Traveler's check 15 = Foreign check
bpRndBasCurHow	126	Integer	Values: 1 = Normal rounding 2 = Always round up 3 = Always round down
bpRndBasCurLvl	127	Boolean	True = round up each item's extended tax amount
bpUseDesc2DefinitionByUDFs	128	Boolean	True = set description #2 field on inventory entry to UDF value
bpDefDesc2ByUDFSAppBlanks	129	Boolean	True = if the UDF value is blank, set the description #2 field to spaces
bpRI_DefaultForNewItems	130	Integer	Values: 0 = non regional inventory 1 = regional inventory



<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpMinSODeposit4CustomerOrder	131	Integer	Minimum sales order deposit amount on customer orders
bpMinSODeposit4SpecialOrder	132	Integer	Minimum sales order deposit amount on special orders
bpMinSODeposit4Layaway	133	Integer	Minimum sales order deposit amount on layaways
bpAllwIntrCompXFer	134	Boolean	True = Allow inter-company transfers
bpUse_Price_Based_Stores	135	Boolean	True = use the price level assigned to that store instead of the currently active price level in inventory
bpBring_Orig_Price	136	Integer	Initial price level on a new POS document detail.
bpACD	137	Boolean	True = use customer price levels
bpQty_Pricing	138	Boolean	True = use quantity price levels
bpBackorderMessage	139	PChar	Back order message text
bpZOutPurgeDays	140	Integer	Purge days for Z-Out information
bpZOutRegEntryOpenAndCloseAmounts	141	Integer	
bpZOutAllowZeroOpenAmount	142	Boolean	True = allow user to specify a zero starting amount
bpZOutAutoCreateNextOpen	143	Boolean	
bpZOutRptCombineSalesTaxVat	144	Boolean	
bpZOutRequireManualCount	145	Boolean	True = require user to specify that they manually counted drawer before continuing

<i>Constant Name</i>	<i>Value</i>	<i>Return Value Datatype</i>	<i>Description</i>
bpZOutRptSortBy	146		
bpWarnCostHigherPrice	147	Boolean	True = warn user that cost is higher than price on inventory item
bpRequireMinPercCustOrder	148	Boolean	True = require deposit on customer orders
bpRequireMinPercSpecOrder	149	Boolean	True = require deposit on special orders
bpRequireMinPercLayawayOrder	150	Boolean	True = require deposit on layaways
bpAccum_Discount	151	Boolean	True = allow cumulative discounts

### ***SbsPreference Constants***

<i>Constant Name</i>	<i>Value (hex)</i>
spClerks	\$00000000
spPass_Word1	\$00000001
spPass_Word2	\$00000002
spPass_Word3	\$00000003
spEncrypt	\$00000004
spResale_No	\$00000005
spFree_PO_Supp	\$00000006
spMargin_Cost	\$00000007
spLook_Ahead_Rows	\$00000008
spItems	\$00000009
spMatrix_Average	\$0000000A
spDefault_Date	\$0000000B
spNo_Companies	\$0000000C
spForm	\$0000000D
spNo_Slip_Security	\$0000000E
spNo_Change_In_Vendor	\$0000000F
spNo_Change_In_Dept	\$00000010

Constant Name	Value (hex)
spVoucher_Trans	\$00000011
sp_PO_Inst1	\$00000012
sp_PO_Inst2	\$00000013
sp_PO_Inst3	\$00000014
sp_PO_Inst4	\$00000015
sp_PO_Inst5	\$00000016
spDont_Use_Last_Cost	\$00000017
spRep_Form	\$00000018
spACD	\$00000019
spCards	\$0000001A
spTransaction	\$0000001B
spDiscount_Explanations	\$0000001C
spQty_Pricing	\$0000001D
spPrice_Desc	\$0000001E
spQty_Decimal	\$0000001F
spRep_Decimal	\$00000020
spAutoGenSlipSetup	\$00000021
spUse_Price_Based_Stores	\$00000022
spB_Secure_Invoice	\$00000023
spB_Secure_Voucher	\$00000024
spBb_Secure_Slip	\$00000025
spMatrix_Definition	\$00000026
spUpdate_Inven_Set	\$00000027
spPickListOrder	\$00000028
spKeep_Original	\$00000029
spAutoUpdateVoucher	\$0000002A
spCurrency_Sym	\$0000002B
spCurrency_Before_Amount	\$0000002C
spRoundTbl	\$0000002D
spUsrFlgMnus	\$0000002E

<i>Constant Name</i>	<i>Value (hex)</i>
spPrecPricTbl	\$0000002F
spTaxCdTbl	\$00000030
spUseVAT	\$00000031
spVatOpts	\$00000032
spTrkIntrn	\$00000033
spZipCodePos	\$00000034
spBasCurNam	\$00000035
spSndOutSlps	\$00000036
spUpdate_Order_Cost	\$00000037
spRequireRecordSale	\$00000038
spOrdrCaseOnly	\$00000039
spPromoPrcFlg	\$0000003A
spUsrCdStrTbl	\$0000003B
spTaxDec	\$0000003C
spTaxAreaNames	\$0000003D
spStore_Code	\$0000003E
spVatPrcDec	\$0000003F
spAllwInpInvcTot	\$00000040
spCoefTyp	\$00000041
spNewCustPolling	\$00000042
spCustUDF	\$00000043
spVendUDF	\$00000044
spDefaultStoreCompany	\$00000045
spTrkAdjMemos	\$00000046
spReqCommAdjMemo	\$00000047
spCenturyDateFormat	\$00000048
spB_Secure_AdjMemo	\$00000049
spAdjMemoReasTbl	\$0000004A
spASNPurgeDays	\$0000004B
spSavStorQtyInAdj	\$0000004C

<i>Constant Name</i>	<i>Value (hex)</i>
spInclTaxMrgP	\$0000004D
spInclTaxMupP	\$0000004E
spInclTaxCoef	\$0000004F
spBUpperCaseReports	\$00000050
spInvnCostMethod	\$00000051
spAllwEdtCustId	\$00000052
pDpsNoteInInvcSO	\$00000053
spNoteFldsOn	\$00000054
spReqPOSFlagFor	\$00000055
spAllwSprdGlobDisc	\$00000056
spUseCurTrngultn	\$00000057
spSegmentedDeptName	\$00000058
spAltGroupLbls	\$00000059
spBUseStoreGroupsInMultiSotrePO	\$0000005A
spAllowVoucherPastPOCancelDate	\$0000005B
spVoucherFileDate	\$0000005C
spAllwDupCustIDs	\$0000005D
spUseTrdDsc4OrdrCst	\$0000005E
spSetpDlrDecCst	\$0000005F
spSetpDlrDecRepCst	\$00000060
spAllwIntrCompXFer	\$00000061
spFilRevDocByFrmrDt	\$00000062
spPollProcDelMrkdAsn	\$00000063
spFldLen	\$00000064
spSubStores	\$00000065
spDefault_Store	\$00000066
spDefault_Station	\$00000067
spReset_Invoice	\$00000068
spPOS	\$00000069
spUse_Shipping_Percent	\$0000006A

<i>Constant Name</i>	<i>Value (hex)</i>
spForeign_Currency	\$0000006B
spTax_On_Shipping_Percent	\$0000006C
spBDefault_Card	\$0000006D
spCheck_Stock_Level	\$0000006E
spRemote	\$0000006F
spIBM_4684_Version	\$00000070
spMultiUser	\$00000071
spClient_Polling	\$00000072
spDefault_Price	\$00000073
spWhat_Price	\$00000074
spShip_Days	\$00000075
spCancel_Days	\$00000076
spRemote_PO	\$00000077
spRemote_SO	\$00000078
spUsesEDI	\$00000079
spAccum_Discount	\$0000007A
spBring_Orig_Price	\$0000007B
spShipping_Percent	\$0000007C
spComnTbl	\$0000007D
spScDisp	\$0000007E
spCheckAvailable	\$0000007F
spShipMethods	\$00000080
spPrintBackorderMessage	\$00000081
spBackorderMessage	\$00000082
spShipFromStore	\$00000083
spShipFromStation	\$00000084
spDefaultShipMethod	\$00000085
spInvnPrcMeth	\$00000086
spInvnPrcAffMan	\$00000087
spMkDnPrcMeth	\$00000088

Constant Name	Value (hex)
spMkDnPrcAffMan	\$00000089
spRelayStore	\$0000008A
spRelayStation	
spNoAllwDelInvcItem	
pReqCustOnRet	
spHiSecSet	
spSOPurgeDays	
spSuggestSC	
spReqSlipComm	
spAllowPOSTaxCodeEdit	
spChkAvSC	
spFrngCurRnd	
spAllwVouPrcUpd	
spCheckAvailOptions	
spNotAllwNegInvcQty	
spTaxRebOk	
spTaxRebPer	
spTaxRebMin	
spDefaultSOInstructions	
spDefaultSOComments	
spPOPurgeDays	
spIgnrMnMxLcksInInvn	
spUseInvcTrms	
spZOutRequireManualCount	
spZOutPurgeDays	
spInvcAllowUpdateOnly	
spDefaultProcessAt	
spLimitItemsAgainstSO	
spCODTenderName	
spRndBasCurlvl	

<i>Constant Name</i>	<i>Value (hex)</i>
spRndBasCurHow	
spShareCustWith	
spDefaultCommCod	
spUseZipCdLkUp	
spZipCdLkUpMsk	
spAutoPollEndPOSInvn	
spMinSODeposit4CustomerOrder	
spMinSODeposit4SpecialOrder	
spMinSODeposit4Layaway	
spReqVchrComm	
spAllwDupALU	
spItmStylSidSrc	
spGenStylSidWthChrs	
spCpyALUFrm	
spUseWsSeqInvc	
spUseWsSeqVchr	
spUseGlbSeqRetInvc	



# Distributing Plugins

## Distributing Plugins

To distribute Plugins, simply copy the Plugin files to the \RetailPro9\Plugins\ folder. Each Plugin typically has at least two files that need to be copied to each client machine that will use the Plugin.

<i>File</i>	<i>Description</i>
[Pluginname].dll	The actual Plugin file.
[Pluginname].mnf	Manifest file.

## Appendix A. Retail Pro Fields and Constants

The following EFT constants are to be used in communicating with Retail Pro. If you find you need a constant that is not present here please contact the Retail Pro development team to implement the change.

### Tender Types

- 0 = Unknown
- 1 = Credit Card
- 2 = Debit Card
- 3 = Gift Card
- 4 = Check
- 5 = EBT

### EFT Result Codes

- 0 = Unknown (EFT Error or Timeout)
- 1 = Success
- 2 = Failed

### CVV2 Result Codes

- 0 = Unknown
- 1 = Match
- 2 = Mismatch
- 3 = Not Processed
- 4 = Missing
- 5 = Not Certified

### AVS Result Codes

- 0 = Unknown
- 1 = Match
- 2 = Address Match Zip Mismatch
- 3 = Zip Match Address Mismatch
- 4 = Mismatch
- 5 = Not Verified
- 6 = Card Not Supported
- 7 = Service Unavailable
- 8 = No Response

## Boolean Constants

RPRO\_TRUE  
RPRO\_FALSE

## EFT Action Codes (Awaiting Revision)

### *Credit*

1 = Sale  
2 = Credit  
3 = Void Sale  
4 = Pre Authorization  
5 = Post Authorization  
6 = Void Credit  
7 = Void Post Authorization  
8 = Commercial Card Sale  
9 = Commercial Card Credit  
10 = Commercial Card Post Authorization  
13 = Gratuity  
14 = Sale with Gratuity  
15 = Book  
17 = Void Book

### *Debit*

41 = Sale  
42 = Return  
43 = Void  
46 = Void Return  
48 = Sale Recovery  
49 = Return Recovery

### *Check*

20 = MICR 20  
21 = COD  
22 = Drivers License  
23 = Double ID  
50 = MICR 50  
54 = Manager Override  
24 = Void Check  
51 = Sale  
52 = Void  
53 = Force  
59 = Settle

## Gift

18 = Balance Inquiry  
 25 = Redemption Sale  
 26 = Register Replace  
 27 = Add Value  
 28 = Activation  
 29 = Void  
 0A = Deactivate  
 0B = Refund  
 0C = Totals Inquiry  
 0D = Previous Day Totals  
 0H = Redemption Unlock  
 0I = Post Authorization  
 0N = Redeem Points  
 0P = Balance Merge  
 0Q = Balance Adjustment  
 0R = Balance Transfer  
 0S = Report Lost  
 0T = Cash Out  
 0U = Add Tip

## Name=Value Name Constants

The following "Name" constants will be used in the AddTender Data parameter.

### Response Fields

RPRO_TenderResult	Tender.
RPRO_Authorization	Number (Required)
RPRO_Reference	Reference Number (If Provided)
RPRO_CVV2Result	Card Verification Value result
RPRO_AVSResult	Address Verification Service result
RPRO_TransID	Gateway Specific Transaction ID (Required)

### ***Credit Request Fields***

RPRO_WorkstationID	Gateway Client ID
RPRO_Amount	DDDDD.CC transaction amount
RPRO_CardNumber	Card Number
RPRO_ExpMonth	MM Expiration Month
RPRO_ExpYear	Expiration Year
RPRO_Keyed Boolean	Boolean indicating if the card was keyed
RPRO_CVV2	CVV2 Number
RPRO_AVS_Street	Cardholder street address
RPRO_AVS_Zip	Cardholder zip code
RPRO_Track2Data	Card Track 2 Data if not Keyed
RPRO_Action	See EFT Action Code list.
RPRO_Ticket_Num	Receipt Number

### ***Debit Request Fields***

RPRO_WorkstationID	Gateway Client ID
RPRO_Amount	DDDDD.CC transaction amount
RPRO_CardNumber	Card Number
RPRO_ExpMonth	MM Expiration Month
RPRO_ExpYear	YY Expiration Year
RPRO_Keyed	Boolean indicating if the card was keyed
RPRO_KeySerialNumer	Debit Key Serial Number
PPRO_PinBlock	Debit Pin block
RPRO_Track2Data	Track 2 Data if not Keyed
RPRO_Action	See EFT Action Code list.

### ***Check Request Fields***

RPRO_WorkstationID	Gateway Client ID
RPRO_Amount	DDDDD.CC transaction amount
RPRO_CardNumber	Account Number
RPRO_Keyed	Boolean indicating if the check was keyed
RPRO_Action	See EFT Action Code list.

RPRO_ZipCode	Check holders Zip Code
RPRO_CashbackAmount	Cash Back amount
RPRO_MICR	MICR Data
RPRO_State	Check holders License State
RPRO_License	Check holders License Number
RPRO_ABANumber	Check ABA Number
RPRO_PhoneNumber	Check holders Phone Number
RPRO_DOB	Check holders Date of Birth
RPRO_CheckNumber	Check Number

### **Gift Request Fields**

RPRO_WorkstationID	Gateway Client ID
RPRO_Action	See EFT Action Code list.
RPRO_Amount:	CC transaction amount
RPRO_CardNumber	Card Number
RPRO_ExpMonth	MM Expiration Month
RPRO_ExpYear	YY Expiration Year
RPRO_Keyed	Boolean indicating if the card was keyed
RPRO_Track2Data	Card Track 2 Data if not Keyed
RPRO_PromoCode	Promotion code
RPRO_CashierNumber	Identifies the cashier.
RPRO_IndustryType	N/A
RPRO_GiftUnits	N/A
RPRO_GiftSequenceNumber	N/A
RPRO_TotalNumberCards	N/A
RPRO_SourceAccountNumber	Account number.
RPRO_ForceFlag	Force authorization flag.
RPRO_PartialRedemptionFlag	Indicates partial redemption of the card value.
RPRO_LoyaltyFlag	Indicates the transaction involves customer loyalty.
RPRO_RefundFlag	Indicates the transaction is a refund.

## Appendix B. Pseudo/Pascal Code Samples

### Simple Sale Transaction

The plug-in would receive the following method calls.

#### *StartTransaction*

```
LogEvent;  
if (FInTrans) or (NOT FEnabled) then exit;  
<<Do Setup Work Here>>  
FInTrans := True;  
Result := True;
```

#### *AddTender*

```
LogEvent;  
Result := -1;  
if (NOT FEnabled) then exit;  
RequestData := Data;  
<<Process Transaction Here>>  
Data := PChar(ResultData);  
Result := Index;
```

#### *CommitTransaction*

```
LogEvent;  
if (NOT FEnabled) or (NOT FInTrans) then exit;  
<<Do Cleanup Work Here>>  
FInTrans := False;  
Result := True;
```

### Sale with a Void Transaction

The plug-in would receive the following method calls

#### *StartTransaction*

```
LogEvent;  
if (FInTrans) or (NOT FEnabled) then exit;  
<<Do Setup Work Here>>  
FInTrans := True;  
Result := True;
```

Retail Pro 9.x EFT Plug-in API

Island Pacific Page 15 of 15

### **AddTender**

```
LogEvent;  
Result := -1;  
if (NOT FEnabled) then exit;  
RequestData := Data; //Contains Name=Value Pairs  
<<Process Transaction Here>>  
Data := PChar(ResultData); //Contains Name=Value Pairs  
Result := Index; //This number used if RemoveTender called
```

### **RemoveTender**

```
LogEvent;  
Result := False;  
if (NOT FEnabled) then exit;  
Result := <<Void Tender based in TenderID>>;
```

### **CommitTransaction**

```
LogEvent;  
if (NOT FEnabled) or (NOT FInTrans) then exit;  
<<Do Cleanup Work Here>>  
FInTrans := False;  
Result := True;
```



# Index

AllBONames .....	134	BOGetRepEntityId .....	98
APIVersion .....	94	BOGetState .....	98
Architecture tips.....	63	BOGetUniqueId .....	101
Base interfaces .....	155	BOIncludeAttrForced.....	122
IAttributeAssignmentPlugin .....	155	BOIncludeAttrIntoInstance .....	123
ICustomAttributePlugin.....	162	BOIncludeAttrIntoList .....	124
IEntityUpdatePlugin.....	155	BOInsert .....	107
IItemAddRemovePlugin .....	158	BOIsAttributeInInstance.....	103
IPrintPlugin .....	162	BOIsAttributeInList.....	103
ISideButtonPlugin .....	161	BOIsEmpty .....	112
ITenderPlugin .....	162	BOIsEntity.....	119
BOCanBeCreated.....	134	BOIsEntityArray .....	120
BOCancel .....	109	BOLast.....	110
BOClear .....	106	BOLocateByAttributes .....	124
BOClearActiveFakeValues .....	119	BOLocatePosition .....	118
BOClearListIncluded .....	104	BONext .....	110
BOClearTempClosingState .....	118	BOOpen .....	106
BOClose .....	106	BOPost.....	108
BOCloseStandAlone .....	107	BOPostAllDataSets .....	114
BOCopy .....	111	BOPostExecute .....	108
BODelete .....	108	BOPreference constants .....	183
BODisableAccesSecurity .....	121	BOPrior .....	111
BODisableDatasetControls .....	115	BORecalculate Attribute .....	119
BODisableDataSetControls .....	113	BORefresh .....	109
BODisableLayoutNotifier.....	116	BORefreshRecords.....	114
BODisableSecurity .....	121	BORollBack.....	111
BOEdit .....	107	BOSetActive .....	<b>101</b>
BOEnableAccesSecurity .....	121	BOSetAttributeValueByName .....	96
BOEnableDatasetControls.....	115	BOSetAttributeValues .....	97
BOEnableDataSetControls .....	113	BOSetCommitOnPost .....	102
BOEnableLayoutNotifier .....	116	BOSetFilter.....	127
BOEnableSecurity.....	122	BOSetFilterAttr .....	126
BOF.....	112	BOSetReadOnly .....	100
BOFirst .....	110	BOSetTempClosingState .....	118
BOFocusAttr .....	122	BOSortOrderByAttribute.....	125
BOGetActive .....	100	BOSortOrderByDomain .....	125
BOGetActiveDatasetId .....	99	BOUpdateActiveInstance .....	113
BOGetAttributeNameList .....	98	BOUpdateDataSetRecords .....	95
BOGetAttributeValueByName .....	96	BOUpdateInstanceCollections.....	104
BOGetAttributeValues .....	97	BOUpdateInstanceDataSet .....	105
BOGetCommitOnPost.....	102	BOUpdateListCollections.....	104
BOGetCopyUniqueId .....	101	BOUpdateListDataSet .....	105
BOGetInstanceActive .....	99	Business object data access.....	89
BOGetPosition.....	117	Business object destruction .....	16
BOGetPref .....	131	Business Object User Interface Events	
BOGetPropValById .....	117	(BOUIE).....	31
BOGetReadOnly .....	100	Business object wrapper handles.....	29

Business object wrappers .....	17	DSUpdateRecord .....	143
TRTIBOWrapper.....	17	Enabled .....	135
Business objects .....	14, 179	Enumerated values.....	179
creation of new business objects..	14	Error codes.....	180
creation of Retail Pro business		Error handling.....	62
objects.....	14	Exceptions.....	63
Button clicks		ExecSQL .....	129
handling .....	35	Exporting procedures.....	57
ChildBOList.....	135	GetAttrPropVal.....	132
CloneBIHandle .....	128	GetAttrPropValues .....	133
COM server object initialization .....	28	GetBOEntityProperties .....	133
Connect .....	128	GetBOTypeForEntityId .....	134
CreateBOByID .....	129	GetHandleForRootBO .....	131
CreateBOByName.....	130	GetPrimaryID .....	148
Creating a new type library .....	38	GetReferenceBOForAttribute .....	132
Creating a Plugin		GetSecurityPermission .....	132
basic steps.....	36	Hardware Plugins .....	67
Creating instances of class factory for		IAbstractPlugin .....	151
each Plugin .....	56	IAttributeAssignmentPlugin .....	155
Creating new COM server object .....	37	IAttributePlugin.....	153
Creating the Plugin classes .....	45	IAttributeValidationPlugin .....	155
Creating the Plugin unit .....	45	IBOPlugin .....	152
Custom Plugin class.....	173	ICustomAttributePlugin .....	162
Data events.....	30	IEFTPlugin .....	164
Development environment setup.....	36	IEntityUpdatePlugin .....	155
Direct data access .....	90	IHardwarePlugin .....	154
Disconnect .....	130	IItemAddRemovePlugin.....	158
Discovery Class		ILicense .....	149
creating .....	53	Importing Plugins.tlb .....	37
method.....	54	Interface implementations.....	70
Distributing Plugins .....	201	Introduction to Plugins.....	7
Document		IPrintPlugin .....	162
events .....	30	ISideButtonPlugin.....	161
DSCreateDataset.....	137	ITenderPlugin .....	162
DSCreateVendor .....	136	Key components .....	10
DSDeleteIndex.....	141	Licensing.....	149
DSDeleteRecord .....	143	Logs .....	64
DSDropDataset .....	138	Manifest files .....	58
DSDropVendor .....	136	Notification/events .....	30
DSGetDatasetSid .....	148	data events.....	30
DSGetRecordByCMSReference .....	147	document events .....	30
DSGetRecordByLookup .....	146	Parent interfaces.....	151
DSGetRecordBySid .....	145	IAbstractBOPlugin .....	152
DSGetRecordSidByCMSReference ..	144	IAbstractPlugin .....	151
DSGetRecordSidByLookup .....	144	IAttributePlugin .....	153
DSGetVendorSID .....	137	IHardwarePlugin .....	154
DSInsertIndex .....	139	Plugin adapters .....	29
DSInsertRecord.....	142	adapter notifications.....	29
DSModifyDataset.....	138	business object wrapper handles ..	29
DSModifyIndexLookup .....	140	Plugin classes	
DSModifyIndexReference .....	140	creating .....	45
DSModifyVendor .....	136	Plugin creation of business objects ..	14

Plugin life cycle .....	10	Sharing child business objects .....	15
Plugin Manager .....	28	TenderBO Wrapper .....	18
Plugin unit		Testing Plugins .....	59
creating .....	45	Tips and tricks .....	63
PluginCapability method .....	95	architecture tips .....	63
Plugins Type Library (Plugins.tlb) ....	91	exceptions .....	63
Preference constants .....	183	logs.....	64
BOPreference .....	183	troubleshooting .....	66
SbsPreference .....	195	Troubleshooting .....	66
Prepared .....	135	TRTIBOWrapper .....	17
Reference.....	135	Type library	
Registering COM servers .....	57	creating .....	38
Regsvr32 .....	58	Types of Plugins .....	8
Retail Pro business object creation ..	14	UpdatePreferences .....	130
Sample uses of Plugins .....	8	User interface data access.....	89
SBSGetPref .....	131	business objects .....	89
SbsPreference constants .....	195	direct data access .....	90