Developer's Guide to Customization

Microsoft Dynamics® AX for Retail POS

January 2011

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics

Table of contents

Introduction

This guide provides instructions and other information for developers who want to extend or customize Microsoft Dynamics® AX for Retail POS. Customizing Retail POS involves the program's interfaces for services and triggers. For details about specific interfaces, see *Interfaces for Services and Triggers*.

Sample code

Sample code for the services and triggers is provided in two downloads, one for partners and one for customers.

- Microsoft Dynamics® AX for Retail POS Plug-ins for Customers
- Microsoft Dynamics® AX for Retail POS Plug-ins for Partners

The downloads are included in the download packages for Microsoft Dynamics AX for Retail POS.

Note

- The plug-in code in the two packages is the same, but the license agreement for use of the code varies for customers and partners.
- Sample code is provided on an as-is basis and is not supported by Microsoft.

Install the sample code

The sample code will be installed to the Retail POS Plug-ins subfolder in the Microsoft Dynamics® AX 2009 installation folder. The typical path to this folder is C:\Program Files\Microsoft Dynamics AX\50\Retail POS Plug-ins.

1. Extract the files in the Retail POS download package to a temporary folder on the hard disk.
2. In the RetailPOSPluginsForPartners or RetailPOSPluginsForCustomers subfolder in the folder where you extracted the files in the Retail POS download package, double-click **AxUpdate.exe**.
3. Complete the Setup Wizard.

Technical background

This guide assumes that the developer is already familiar with C#, the .NET framework, and Microsoft Visual Studio 2008.

Localization

Retail POS has been localized for many different countries. The following values might affect customizations.

Value	Details
CurrentUICulture	This value (System.Threading.Thread.CurrentThread.CurrentUICulture) is set during application start. It identifies the culture used by the program. It is configured in Microsoft Dynamics AX. The value comes from CULTURENAME in the RBOSTORETABLE unless there is a specific culture set for the cashier. In that case, it comes from

Value	Details
	OPERATORCULTURE in the RBOSTAFFTABLE.
CurrentCulture	This value (System.Globalization.CultureInfo.CurrentCulture) is the default for the system configuration. It is generally used for formatting data (such as dates and numbers).
StoreCurrency	This value (LSRetailPosis.Settings.ApplicationSettings.Terminal.StoreCurrency) is identifies the default store currency.
CompanyCurrency	This value, like StoreCurrency, identifies the default company currency.

Splash screen service tutorial

This tutorial describes creating a custom splash screen service. You can use this to be customized service splash screen (the screen shown during the application startup sequence). At the end of this tutorial one should be able to create a new service, implement some of its features, debug issues, and activate the service.

Note

Before starting this tutorial, backup the installed splash screen service, in case you need to restore it. Save the backup in the **Service** folder where Microsoft Dynamics® AX for Retail POS is installed.

Create the project

The first step in creating our custom splash screen will be to create a new Visual Studio project and to configure the desired settings and add the appropriate references.

1. Start Visual Studio.
2. On the **File** menu, click **New**, and then click **Project**.
3. In the new project window, under **Visual C#** select **Class Library**. Give the project (and solution) a name such as "SplashScreen" and select the location where you wish the project to be created.

   Note

   You may provide a different name, but for the SpashScreen service to be recognized, the DLL and namespace must be "SplashScreen".

4. In the **Solution Explorer**, right-click **SplashScreen**, and then select **Properties**.
   a. On the **Application** tab, make sure the **Assembly** name is **SplashScreen** and the output type is **Class Library**.
   b. On the **Build** tab, change the output path to the **Services** folder in the folder where Retail POS is installed.
   c. On the **Debug** tab, set **Start Action** to **Start external program**, and then select Retail POS.
   d. Services and triggers must be signed, so on the **Signing** tab, select **Sign the assembly**.

      You can use your company's signature file, or for the purpose of this tutorial, create a new one (with an empty password), such as "SplashScreenTutorial."

   e. On the **Code Analysis** tab, select **Enable Code Analysis on Build**.

      This activates Code Analysis to make sure .NET guidelines are followed.

      Optionally, you can select **Treat Warning as Error** for each category of rules.
5. Mark the assembly as CLS Compliant. Click the "+" sign next to the **Properties** folder in the **SplashScreen** project. Double-click **AssemblyInfo.cs** to open it.
   a. Add "`using System;`" to the top of the file.
   b. Add the CLS compliant attribute: "`[assembly: CLSCompliant(true)]`" to the bottom of the file.
6. Verify that you can build the project without errors. On the **Build** menu, click **Build Solution**.

Implement the splash screen

1. Rename **Class.cs** in Solution Explorer to **SplashScreen.cs**.
2. When prompted to rename all reference to the code element **Class1**, click **Yes**.

   You should now have a class named **SplashScreen** in the namespace **SplashScreen**.
3. Right-click **References**, and then select **Add References**.
4. Click the Browse tab, open the Retail POS installation folder, and then select **SplashScreenInterface.dll**.
5. Add a form for the splash screen:
   a. In **Solution Explorer**, right-click the project, click **Add**, and then select **Windows Forms**.
   b. Create a Windows form with the name "FormSplashScreen" and modify the form as desired. For example, set a background image, size, start position, disable **ShowIcon**, disable **ShowInTaskbar**, or set the form boarder style to **None**.
   c. Add a label to the form and title it "Splash Screen Tutorial."
   d. Optionally, you can override form events, such as **FormClosed**.
6. Modify **SplashScreen.cs** by adding the following using statement: "`using SplashScreenInterface;`".
7. Modify the class so that it implements the "`ISplashScreen`" interface. Note that this interface contains three methods:
   a. Implement **CloseSplashScreen()** as follows:

```
if (splash != null)
{
   splash.Close();
}
```

   b. Implement **DisplaySplashScreen()** as follows:

```
splash = new FormSplashScreen();
splash.Show();
splash.Refresh();
```

   c. Implement **UpdateSplashScreen()** as follows:

```
if (splash != null)
{
   splash.Refresh();
}
```

8. Compile the solution.

   Three warnings are displayed.
9. For the purposes of this example, suppress the warnings by doing the following.
   a. CA1824 – Right-click the message, and then select **Suppress in project suppression file**.
   b. CA1724 – Right-click the message, and then select **Suppress in source**. You must suppress this for all services and triggers.
   c. CA1001 – Right-click the message, and then select **Suppress in source**. The ISplashScreen implements "CloseSplashScreen" which acts as the dispose method.

Verify and debug the splash screen service

This section describes running and debugging the service. Recall that the project file was set to run Retail POS. The DLL that was created was placed in the service folder in the Retail POS installation folder. This ensures that Retail POS runs when debugging and that the service is available to Retail POS. Also, the service was signed, as this is a requirement for services.

Verify the service

- On the **Debug** menu, click **Start Debugging**.

  -or-

- Press F5.

  The splash screen with the custom text “Splash Screen Tutorial” should be displayed momentarily just before the logon screen.

Debug the service

1. Set a breakpoint in “DisplaySplashScreen” on the line: “`splash = new FormSplashScreen();`”.
2. Press F5 to run the debugger.
3. The program should start but quickly hit the breakpoint. It will be highlighted in the debugger.
4. Add a breakpoint in the “CloseSplashScreen()” method.
5. Press F5.

   The breakpoint in the close method is reached. “splash” should not be null.
6. Press F10 (**Debug** > **Step Over**) until reaching the splash.Close() method.
7. Press F5 to continue.
8. The application should reach the logon screen.

Fiscal framework tutorial

In this section, we illustrate a sample integration for a fictitious fiscal printer.

Fiscal printers are used in various countries for auditing and tax purposes. They are a distinct device and should not be confused with a receipt printer.

As the rules and regulations for fiscal printers vary significantly from country to country, this example includes various rules from several different countries. This example does not implement all of the rules for any particular country or region.

Overview

The requirements for this integration example are:

- All sales transactions have a fiscal receipt coupon. Other transaction types are not covered by this tutorial.
- On startup, Retail POS must validate that the fiscal printer was the same one used the last time based on the grand total and Z report data. If the grand total is not a match, then the Z-Report is validated. If the Z-report is not a match, Retail POS asks if you want to make this the new fiscal printer. If not, Retail POS closes.
- Persist printer grand total, serial #, Z-report.
- Show a custom splash screen on startup.
- The fiscal printer is not able to apply discounts to a line item once.
- The fiscal printer is not able to adjust quantity for a line item.
- Transaction level discounts can only be applied one time, and only after all line items have been added to the printer.
- Printer is configured with three tenders:
    - “Dinheiro”
    - “Visa Credito”
    - “Visa Debito”
- The printer is configured with four taxes:
    - TA = 12.00%
    - TB = 5.0%
    - TC = 5.0%
    - TD = 10.0%
- After each receipt coupon is completed, the grand total should be saved.
- Returns are not allowed.
- Transaction voids cancel the last or current fiscal receipt coupon.
- Line item voids are allowed.
- Compute and persist POS.EXE MD5 hash.
- A Fiscal Printer menu shall be supported, that implements at least the following:
    - Show printer grand total

    - Show printer subtotal
    - Generate a Z-report (using built-in printer report)
    - Generate an X-report (using built-in printer report)
    - Generate a customer management (non-fiscal) report
    - Show built-in printer tax rates
    - Show current/next receipt coupon #
    - Compute MD5 hash value for all files (.exe, .dll) in the program and sub-folder and attempt to save to the program folder.
- Menu option to show Item Price list (base sale price is sufficient).
- When sales items are added to the transaction it should be printed on the fiscal printer, and must include the following data (whole quantity is sufficient for this sample):
    - Retail Price (without discounts)
    - Description
    - Item # (e.g., Item lookup code)
    - Tax (using printer Tax Code)
    - Quantity
- When posting, include the printer serial # and fiscal receipt coupon #.
- Request customer tax # when starting a fiscal receipt coupon.
- Print management report when non-cash tenders are include in the transaction (after the receipt coupon fiscal is printed).

Setup

This sample requires that a copy of the POS application program folder is located at the following location relative to the Tutorial folder:

..\LSPOSNET\POS\bin\Debug

Before getting started, copy the files to this location (a setup.bat file is provided that may be customized if desired to copy the files).

Solution project overview

The FiscalSample will consist of a number of projects which are dropped in various locations (the POS applications root folder, or the triggers or service sub folder). All projects have code analysis rules active and set to cause build errors.

Project	Drop Location	Overview
ApplicationTriggers	%posroot%\triggers	Overrides application triggers (such as startup to initialize the printer)
BlankOperations	%posroot%\services	Provides additional function keys such as for displaying a fiscal menu

Project	Drop Location	Overview
FiscalCore	%posroot%	Provides core integration between the various components
FiscalLog	%posroot%	Provides core logging
FiscalPrinterInterface	%posroot%	Provides generic fiscal printer interface. Abstracts the interface so that custom code for the printer is isolated from the rest of the example.
ItemTriggers	%posroot%\triggers	Provides functions for interfacing with a fiscal printer as items are added, removed, etc.
SimulatedFiscalPrinter	%posroot%	Provides a UI that simulates a fiscal printer.
SplashScreen	%posroot%\services	Customized application splash screen
TransactionTriggers	%posroot%\triggers	Provides fiscal printer operations for completing a transaction.

Generic fiscal printer interface

Because the API provided by each Independent Hardware Vender (IHV) who manufactures Fiscal Printers, we are first going to provide an abstraction to capture some of the primary operations that we expect our fictitious fiscal printer to support. The idea here is that this interface should be easily adaptable to many existing fiscal printers.

The purpose of this class is to provide a generic interface between the fiscal sample and the physical device. This interface provides an abstraction that can then be implemented to meet the needs of a given physical fiscal printer and its own API requirements. Because this class is intended to provide a generic interface with fiscal printers it focuses on the core operations related to fiscal receipt coupons. In production code it is assumed that this interface may need to be extended to meet specific local or printer needs.

This project, “FiscalPrinterInterface” will provide the create the generic abstraction used by the application (specifically FiscalCore, see below) to interface with a physical fiscal printer. The actual implementation of this interface will be vendor specific for any given physical fiscal printer.

The fiscal printer interface will include a definition for the following key elements:

Class/Interface/Enum	Detail
BarcodeTextPosition	Enumeration of common text positions associated with barcodes.
FiscalBarcodeSymbology	Enumeration of the most common barcode symbolizes supported by fiscal printers.
FiscalPrinterState	Enumeration of the supported printer states: • None – Instance object of IFiscalOperations has not yet be initialized or is in an unknown state. • Open – IFiscalOperations has opened a connection to the

Class/Interface/Enum	Detail
	printer • FiscalReceipt – In receipt coupon mode • ManagementReport – In non-fiscal management report mode.
IFiscalOperations	Provides a generic interface for working with Fiscal Printers

In production code, it would likely be a good idea to also define a generic Fiscal Printer exception that can be used when device errors occur; however, for simplicity in this example we will not be doing this.

The main methods provided by the IFiscalOperation interface include:

Method	Detail
AddItem	Adds an item to the fiscal printer and returns the printer line item number NOTE: valid after BeginReceipt and before StartTotalPayment
ApplyDiscount	Apply an item discount to the printer NOTE: valid after BeginReceipt and before StartTotalPayment
BeginReceipt	Begin a coupon fiscal operation
CancelReceipt	Cancel the coupon fiscal operation
EndReceipt	End the fiscal coupon receipt
GetBalanceDue	Get the balance due NOTE: valid after StartTotalPayment
GetChangeDue	Get the change due to the customer from the fiscal printer NOTE: valid after EndReceipt.
GetCouponNumber	Get the current coupon # from the fiscal printer
GetGrandTotal	Get the grand total from the fiscal printer
GetSerialNumber	Get the serial number of the fiscal printer
GetSubtotal	Get the current subtotal from the fiscal printer
GetTaxRates	Get the set of tax rates in the printer. This includes the Tax Code Identifier (e.g., “TA”, “TB”, …) and the tax rate (percentage)
Initialize	Initialize the fiscal printer object
MakePayment	Make a payment (valid after StartTotalPayment) NOTE: valid after StartTotalPayment and before

Method	Detail
	EndReceipt
ManagementReportBegin	Starts a management report
ManagementReportEnd	Ends the management report
ManagementReportPrintBarcode	Prints a barcode on the management report
ManagementReportPrintLine	Prints a line on a management report
Open	Open a connection to the fiscal printer
OperatingState	Returns the fiscal printers current state
PrintXReport	Print the built-in X-report on the fiscal printer
PrintZReport	Print the built-in Z-report on the fiscal printer. NOTE: Once done the books are closed for the day and the fiscal printer will not accept any more sales transactions.
RemoveItem	Removes the specified line item NOTE: valid after BeginReceipt and before StartTotalPayment
StartTotalPayment	Start the total payment. Optional APIs includes with transaction level discount or surcharge.
TryReadLastZReport	Get a digital copy of the last Z-report printed by the fiscal printer.

Fiscal core

This is the central location for all common operations related to the fiscal printer. It is placed in the Retail POS root folder as this allows triggers and services to provide a way to interact that otherwise they would not have.

The main class is a singleton that provides access to the fiscal printer object, which implements the IFiscalOperations interface. The singleton is a full lazy singleton class which means that one and only one instance can ever exist and that the instance is not created until it is first used. From that point onward, the one singleton is available both to services and triggers.

In order for both services and triggers to access this singleton, it must be placed in the application root folder so that any sub-folders under POS (such as triggers and services) are able to access the class.

Class	Description
CustomerTaxIdForm	This form is used to get request customer tax ID at the start of a transactions.
FiscalPrinterSingleton	Provides a singleton fiscal printer access and operations. Also functions as a factory for the PrinterFiscalInterface.
GlobalSuppressions	Project Code Analysis suppressions

Class	Description
LineItemTagalong	A container class that holds additional data needed for fiscal operations associated with a line item.
PersistentPrinterData	A serializable class used to persist data associated with the fiscal printer.
UserMessages	A utility class used by triggers and services to display forms on the UI

The Fiscal Printer Singleton provides the flowing operations

Method	Description
ComputeFileHash	Compute the MD5 has for a given file. Opens the file as read-only to perform the computation on files such as on the running POS.exe.
Ctor	Constructor. Initializes data, including mapping.
DynamicLoadFiscalPrinter	Dynamically loads the class that implements the fiscal printer interface. Defaults to the "SimulatedFiscalPrinter" if no override is provided.
FindTaxRateId	Find the mapping tax rate identifier. Attempts to map this to the printers tax rate. Note: Will return "TA" if a match is not found.
FiscalCorePath	Returns the path to this DLL.
FiscalPrinter	Gets the fiscal printer
GetPosExeMD5	Get the computed MD5 for the POS.exe application.
Instance	Return the single instance of this class
MapTendertypeIdToPaymentMethod	Attempts to find the mapping of the tender type ID to the printer.
PrinterData	Gets the persistent printer data
ProgramFileList	Returns a list of all EXE and DLL files along with the computed MD5 hash value.
SalesLineItemData	Collection of tag-along data for sales line items. Mapped using the POS sales line item number.
SaveProgramListMd5	Attempts to saves the MD5 data for the program list into the same folder as the POS.exe.
SetTaxRateFromPrinter	Sets the _taxRate data as obtained from the printer
UpdateFiscalCouponSalesItemQty	Checks for quantity changes to the retailTranscation sales line items. If found, the change is made to the printer and associated tag along data. This includes adding a new item. NOTE: Discounts are held back until the end of

Method	Description
	transaction as some printers only allow discounts to be applied one time yet the POS may make many changes due to discounts.

LineItemTagalong is a data class that contains the following:

Method, Property	Description
Ctor	Constructor.
PostedPrice	Posted price
PostedQuantity	Last posted quantity on the fiscal printer
PrinterItemNumber	Printer line item number
TaxRateId	Tax rate ID given to the fiscal printer for the line item
Voided	Was the line item voided

Persistent data

The following classes are used for persistent data access

Class	Description
PersistenPrinterDataBinder	Implements a SerializationBinder. Required so that the class can be deserialized from triggers and services which run under their own context in the sub-folder below the POS application folder.
PersistentHelp	A helper class used to persist objects to and from files.
PersistentPrinterData	A serializable class used to persist data associated with the fiscal printer.

PersistenPrinterData is an ISerializabel class used to persist key fiscal printer data. This data is used, for example, on application start to verify that the printer has not changed. It implements the following:

Method, Property	Description
Ctor	Default ctor
Dirty	Flag indicates class is “dirty”. That is it has been changed but not saved to file yet.
GrandTotal	Fiscal printer Grand Total
PersistedObject	Gets the class from a file. Default if not able to restore from file.
PersistenPrinterData	Static class constructor
SerialNumber	Fiscal printer serial number
SetGrandTotal	Set the grand total and persist to file.

Method, Property	Description
SetSerialNumber	Sets the serial number and persist to file.
SetZReport	Set the Z-Report and persist to file.
ZReport	Fiscal printer Z-Report Data

Mappings

In this example, the Tender IDs are mapped to the following printer tenders:

Tender ID	Printer Tender
1	Dinheiro
2	Visa Credito
3	Visa Debito

The taxes in this example are mapped as follows:

Tax Code (AX)	Printer Tax Code	Printer Tax Rate
AV_DCST	TA	12.00%
AV_MDST	TB	5.0%
SP_DCST	TC	5.0%
NA	TD	10.0%

Fiscal log

This class demonstrates how to provide common functionality that can be accessed by both triggers and services (similar to Fiscal Core). It simply provides logging capability. In production code one may wish to use this as a starting point for implementing code that logs to the event log.

This class implements the singleton design pattern. This singleton is accessible by both services and triggers. In order for both services and triggers to access this singleton, it must be placed in the application root folder so that any sub-folders under POS (such as triggers and services) are able to access the class.

Application triggers

Application Triggers implementation for the fiscal printer operations. Because this is an implementation of the application trigger it must implement the IApplicationriggers class, this class (and DLL) must be named “ApplicationTriggers” and the DLL must be placed in the triggers sub-folder under the application.

Override	Description
ApplicationStart	Used to initialized and open the fiscal printer. It then checks if the grand total and serial # match the last saved value from the application, if they don’t match, the last z-report is

Override	Description
	checked to see if it is the same. If not the user is prompted to quit or update to the new printer. Next, we get the tax info from the fiscal printer. Finally, we check to see if the fiscal printer has a sub-total, if so we cancel the receipt coupon.
ApplicationStop	Logged only
LoginWindowVisible	Logged only
Logoff	Logged only
PostLogon	Logged only
PreLogon	Logged only

Item triggers

Item Triggers implementation for the fiscal printer operations. Because this is an implementation of the application items trigger it must implement the IItemTriggers class, this class (and DLL) must be named "ItemTriggers" and the DLL must be placed in the triggers sub-folder under the application.

Override	Description
PostSale	This method is where most of the action is done. This method with check to see if the posTranscation is a RetailTransaction and if any SaleLineItems have been added to the transaction. If so, it will perform the following: Start a fiscal receipt coupon (if one has not yet been started) Check to see if item # has been processed, if not, update the sales item on the printer and in the fiscal core tag-along NOTE: If item has been processed already, we will make quantity or discount changes later. Note: Depending on application needs, one may wish to skip this processing for certain types of sales line items, such as Income or Expense account.
PostSetQty	This API is not used in this example.
PostVoidItem	Remove a voided sales item from the fiscal printer receipt coupon.
PrePriceOverride	Prohibited in this example. While discounts will be allowed, we will not allow price overrides. So a standard error message is displayed for the user.
PreReturnItem	Prohibited in this example. We will display a standard error message.
PreSale	This API is not used in this example
PreSetQty	This API is not used in this example

Transaction triggers

Transaction Triggers implementation for the fiscal printer operations. Because this is an implementation of the application transaction trigger it must implement the ITraansactionTriggers class, this class (and DLL) must be named "TransactionTriggers" and the DLL must be placed in the triggers sub-folder under the application.

Override	Description
BeginTransaction	Operation logged only
PostEndTransaction	Logged only
PostReturnTranscation	Logged only
PostVoidTransaction	Void operations result in canceling the receipt coupon.
PreEndTranscation	Main posting logic for fiscal printer. Performs the following scenario logic: Will compute the POS.EXE MD5 value if it has not already been computed. Check to see if a retail Transaction and that the fiscal printer is in the FiscalReceipt state Updates any quantity changes on the fiscal printer. Apply all line-level discounts (amount and percent) on the fiscal printer Determine one total transactional discount (amount or percent) Start payments and apply total discount (amount or percentage) For each tender, map to the printer tender and apply the tender amount on the fiscal printer. Verify that sufficient funds have been provided for the fiscal receipt coupon End the fiscal receipt coupon Updated the persisted Grand Total for the printer. If an error is detected the user is informed and operation is canceled.
PreReturnTransaction	Operation prohibited in this example. A standard dialog is displayed to the user.
PreVoidTransaction	Logged only

Blank operations service

Blank operations service implementation for the fiscal printer operations. Because this is an implementation of the blank operation service it must implement the IBlankOperations class, this class (and DLL) must be named "BlankOperations" and the DLL must be placed in the services sub-folder under the application.

Class	Description
BlankOperations	The service entry point. Demonstrates how to create a blank operation. This includes how to get access to the SQL connection along with key data. This implementation provides the following operation ids: 1 – Display a general fiscal menu 2 – display item price list
GeneralFiscalForm	The general “Fiscal Menu” form.
GlobalSuppressions	Global Code Analysis suppressions
ItemPriceListForm	Item Price List form – Displays an item price list. Demonstrates how to access the DB to get information such as item prices, as well as how to get the store id, terminal id, and data read id.

The BlankOperations class implements the following:

Override	Description
BlankOperation	Implements the logic that displays the operations based upon the provided operation identifier. Default if no operation identifier is provided is to display the Fiscal Menu.
Ctor	Provide a constructor that takes in a Sql Connection and the dataReadId. Saves these values for access to the SQL database.

The General Fiscal Form demonstrates how to perform the following operations:

Operation	Description
Change Due	Get the change due for the last receipt coupon
Get Coupon #	Get receipt coupon #
Grand Total	Get the grand total from the fiscal printer
Management Report	Print a custom management report
MD5 Computation	Save all MD5 computed data for program and DLLs into the program folder.
Subtotal	Get the printers subtotal value for the current receipt coupon
Tax Rates	Shows configured printer tax rates.
X Report	Print an X report using the built in printer report
Z Report	Print a Z report using the built in printer report
Z Report Data	Get the Z report from the data from the printer and persist this into a file.

Splash screen service

Splash screen service implementation for the fiscal printer operations. Because this is an implementation of the splash screen service it must implement the ISplashScreen class, this class (and DLL) must be named “SplashScreen” and the DLL must be placed in the services sub-folder under the application.

Override	Description
CloseSplashScreen	Close the splash screen (if it was created)
DisplaySplashScreen	Displays the splash screen
UpdateSplashScreen	Causes the splash screen to refresh

Simulated fiscal printer

Provides a sample implementation of the Fiscal Printer Interface (IFiscalOperations). This implementation emulates a fiscal printer on the screen. One is expected to provide their own implementation that interfaces with the fiscal printer for their own region.

Class	Description
GlobalSuppression	Global code analysis suppression file
LineItem	LineItem data used by the emulator
MyFiscalPrinter	Implements the Fiscal Printer Interface. Uses the SimulatedFiscalPrinterForm to act as the fiscal printer
SimulatedFiscalPrinterForm	Emulator for a fiscal printer
TaxInfo	Maintains tax info for the emulator
ZReport	Maintains Z-Report data for the emulator

Appendix

Services and triggers are loaded by file name. For Retail POS to recognize customized services and triggers, they must use the same file names as the services and triggers that they replace.

The service and trigger file names associated with the interfaces are listed in the following tables.

Service interface	Service file name
IApplication interface	Application.dll
IBarcodes interface	Barcodes.dll
IBlankOperations interface	BlankOperations.dll
ICard interface	Card.dll
ICashChanger interface	CashChanger.dll
ICCTV interface	CCTV.dll
ICorporateCard interface	CorporateCard.dll
ICreateDatabase interface	CreateDatabase.dll
ICreditMemo interface	CreditMemo.dll
ICurrency interface	Currency.dll
ICustomer interface	Customer.dll
IDialog interface	Dialog.dll
IDimension interface	Dimension.dll
IDiscount interface	Discount.dll
IDualDisplay interface	DualDisplay.dll
IEFT interface	EFT.dll
IEOD interface	EOD.dll
IGiftCard interface	GiftCard.dll
IItem interface	Item.dll
ILoyalty interface	Loyalty.dll
IPrice interface	Price.dll
IPrinting interface	Printing.dll
IPurchaseOrder interface	PurchaseOrderReceiving.dll
IRFID interface	RFID.dll
IRounding interface	Rounding.dll
ISalesInvoice interface	SalesInvoice.dll
ISalesOrder interface	SalesOrder.dll
IStockCount interface	StockCount.dll

Service interface	Service file name
IStoreInventoryServices interface	StoreInventoryServices.dll
ITax interface	Tax.dll
ITenderRestriction interface	TenderRestriction.dll

Trigger interface	Trigger file name
IApplicationTriggers interface	ApplicationTriggers.dll
ICashManagementTriggers interface	CashManagementTriggers.dll
ICustomerTriggers interface	CustomerTriggers.dll
IDiscountTriggers interface	DiscountTriggers.dll
IInfocodeTriggers interface	InfocodeTriggers.dll
IItemTriggers interface	ItemTriggers.dll
IPaymentTriggers interface	PaymentTriggers.dll
ISuspendTriggers interface	SuspendTriggers.dll
ITransactionTriggers interface	TransactionTriggers.dll