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

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Copyright © 2011 Microsoft. All rights reserved.

Microsoft, Microsoft Dynamics, and the Microsoft Dynamics Logo are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.
# Table of contents

Introduction .................................................................................................................. 1  
Sample code .................................................................................................................. 1  
Interfaces for services ................................................................................................. 1  
Interfaces for triggers ................................................................................................. 2  

## Interfaces for services

- IApplication interface .............................................................................................. 3  
- IBarcodes interface .................................................................................................. 3  
- IBankOperations interface ....................................................................................... 4  
- ICard interface ......................................................................................................... 4  
- ICashChanger interface ........................................................................................... 5  
- ICashDrawer interface .............................................................................................. 7  
- ICCCTV interface ..................................................................................................... 7  
- ICorporateCard interface ......................................................................................... 8  
- ICreateDatabase interface ...................................................................................... 9  
- ICreditMemo interface ............................................................................................. 9  
- ICurrency interface .................................................................................................. 11  
- ICustomer interface ................................................................................................. 11  
- IDialog interface ...................................................................................................... 13  
- IDimension interface ............................................................................................... 15  
- IDiscount interface ................................................................................................. 15  
- IDualDisplay interface ............................................................................................. 17  
- IEFT interface ......................................................................................................... 17  
- IEOD interface ......................................................................................................... 19  
- IGiftCard interface .................................................................................................. 20  
- IInfocodes interface ............................................................................................... 21  
- IItem interface ........................................................................................................ 23  
- IKeyLock interface ................................................................................................. 23  
- ILineDisplay interface ............................................................................................ 23  
- ILoyalty interface ..................................................................................................... 24  
- IMSR interface ....................................................................................................... 25  
- IPeripherals interface ............................................................................................. 25  
- IPinPad interface ..................................................................................................... 25  
- IPrice interface ....................................................................................................... 25  
- IPrinter interface ..................................................................................................... 26  
- IPrinting interface ................................................................................................... 26  
- IPurchaseOrder interface ....................................................................................... 29  
- IRFID interface ...................................................................................................... 29  
- IRFIDScanner interface ........................................................................................... 30  
- IRounding interface ............................................................................................... 30  
- ISalesInvoice interface ......................................................................................... 30  
- ISalesOrder interface .............................................................................................. 30  
- IScale interface ....................................................................................................... 31  
- IScanner interface ................................................................................................... 31  
- ISStockCount interface ......................................................................................... 31  
- IStoreInventoryServices interface ....................................................................... 31  
- ITax interface ......................................................................................................... 31  
- ITaxProvider interface ........................................................................................... 33  
- ITenderRestriction interface .................................................................................. 34
# Table of contents

## Interfaces for peripherals
- ICashDrawer interface ................................................................. 35
- IDualDisplay interface ................................................................. 35
- IKeyLock interface ........................................................................ 35
- ILineDisplay interface ................................................................. 35
- IMSR interface .............................................................................. 37
- IPeripherals interface ................................................................. 37
- IPinPad interface ........................................................................... 37
- IPrinter interface .......................................................................... 38
- IRFIDScanner interface ................................................................. 38
- IScale interface ............................................................................. 39
- IScanner interface ......................................................................... 39

## Interfaces for triggers
- IApplicationTriggers interface ..................................................... 40
- ICashManagementTriggers interface ............................................. 41
- ICustomerTriggers interface ......................................................... 41
- IDiscountTriggers interface .......................................................... 41
- IInfocodeTriggers interface .......................................................... 42
- IItemTriggers interface ................................................................. 42
- IPaymentTriggers interface ............................................................ 44
- ISuspendTriggers interface ............................................................ 44
- ITransactionTriggers interface ..................................................... 46

## Appendix
- Appendix ............................................................................................ 48
Introduction

This document is a reference for developers who want to use the interfaces for services and triggers in Microsoft Dynamics® AX for Retail POS to customize and extend many of the program's features.

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.

Interfaces for services

Many features in Retail POS are services that interact with the program through interfaces. Retail POS loads the services at run time and calls them through the interfaces whenever certain information or processes are needed. Services can be customized so that you can extend features in Retail POS, such as calculating taxes or communicating with electronic funds transfer (EFT) providers. For descriptions of all the service interfaces that are available in Retail POS, see "Interfaces for services" later in this guide.

Although most services provide specific features, a Blank service is also provided. You can use this service to implement custom operations, such as interacting with third-party software. This service can be invoked from the Blank operation or from any other operation in Retail POS.

When you customize a service, you must keep the same file name as the original; otherwise, Retail POS will not recognize it. You can find a list of these file names in the Appendix.
Interfaces for triggers

Triggers are called by Retail POS before and after various operations. You can use triggers to add customized code before (pre-trigger) and after (post-trigger) specific operations. Pre-triggers provide an extra layer of verification and can return *false* if the operation should be canceled. Post-triggers provide a way to respond to an operation after it has finished. Triggers can be extended in the same manner as services. For descriptions of all the triggers interfaces that are available in Retail POS, see "Interfaces for triggers" later in this guide.
Interfaces for services

IApplication interface

The IApplication interface enables custom receipt numbering formats.

Methods

- string **GetNextReceiptId** (string receiptIdNumberSequence)
  Returns a Receipt ID that is based on a seed coming from Retail Headquarters. If no seed information is provided, this method uses the highest receipt number available based on a custom algorithm.

  This method is triggered when a new sale is started.

  Parameters
  - receiptIdNumberSequence – The number sequence ID used to create the receipt IDs

- void **IncrementReceiptId** (string receiptIdNumberSequence)
  This method increments the information with the next receipt number if number sequence is used.

  This method is triggered when a new sale is saved to the database.

  Parameters
  - receiptIdNumberSequence – The number sequence ID used to create the receipt IDs

IBarcodes interface

The IBarcodes interface processes the original bar code to populate the different bar code properties in the BarcodeInfo class.

Methods

- BarcodeInfo **ProcessBarcode** (ref SaleLineItem saleLineItem, ScanInfo scanInfo, BarcodeInfo barcodeInfo, string barcode)
  Processes the scanned bar code (thereby searches the database) and returns an instance of BarcodeInfo.

  This method is triggered when a bar code item is scanned.

  Parameters
  - saleLineItem (out) – The sales line item for populating item information based on the bar code
  - scanInfo – ScanInfo class object
  - barcodeInfo – BarcodeInfo class object
  - barcode – The scanned bar code

- BarcodeInfo **ProcessBarcode** (BarcodeEntryType entrytype, string barcode)
  Currently not used.
• BarcodeInfo ProcessBarcode(ScanInfo scanInfo)
  Processes the scanned bar code (thereby searching the database) and returns an instance of BarCodeInfo.
  This method is triggered when a bar code item is scanned.
  Parameters
  o scanInfo – ScanInfo object

IBlankOperations interface
The IBlankOperations interface enables Retail POS to be extended with new operations.
Many different Blank operations can be referenced from within the Blank operation service. When executing a Blank operation, the Blank operation service is called with the appropriate operation number.

Methods
• void BlankOperation(ref BlankOperationInfo operationInfo, ref PosTransaction posTransaction)
  Handles the custom operation.
  This method is triggered when executing a blank operation.
  Parameters
  o operationInfo – Operation data (for example, a custom operation number)
  o posTransaction – Transaction object

ICard interface
The ICard interface provides methods to perform validity checks when a card is used for payment, either by swiping a card or by manually entering the card information. The main function of the ICard interface is to identify the card type and send its information to the appropriate service, such as electronic funds transfer (EFT), corporate card, or loyalty card service.

Methods
• void GetCardType(ref CardInfo cardInfo)
  Identifies the type of the card and returns one of the types listed in "Card types" later in this topic.
  This method is triggered whenever a card is swiped or card information is manually entered.
  Parameters
  o cardInfo – The detailed card information
• bool IsCardLengthValid(string cardNumber)
  Used when the cashier manually enters the card number.
  Parameters
  o cardNumber – The number of the card
• bool IsExpirydateValid(string expirationDate)
  Used only when the cashier manually enters the expiration date.

  Parameters
  o expirationDate – The expiration date of the card

Card types
• International Credit Card
  When a card is identified as a credit card, the ProcesscardPayment method is called in the EFT service (EFT.dll). This method connects to an external card broker that authorizes or rejects the card. If the card is authorized, Retail POS adds a card payment line to the transaction.

• International Debit Card
  When a card is identified as a debit card, the ProcesscardPayment method is called in the EFT service (EFT.dll). This method connects to an external card broker that authorizes or rejects the card. If the card is authorized, Retail POS adds a card payment line to the transaction.

• Unknown Card
  If the card service does not identify the card, the IdentifyCard method in the EFT service is called to identify the card. Then ProcessCardPayment is called in the EFT service.

• Corporate Card
  A corporate card is issued and authorized by the company using Retail POS. This can be a fleet card or a regular corporate card.

• Loyalty Card

• Customer Card
  A customer card identifies the customer but is not used for payments, as the corporate card is. The card adds customer information to the transaction.

• Employee Card
  Employees use an employee card to log on to Retail POS.

• Salesperson Card
  A salesperson card adds a salesperson to the transaction.

ICashChanger interface
The ICashChanger interface provides utility methods to communicate with the cash changer device.

Methods
• bool Initialize()
  Initializes the cash changer hardware.
  This method is triggered when the Cash Change dialog box is displayed.

  Parameters
  o text – Message text
• bool **Login**(string *terminalID*, string *operatorID*)
  Logs on the current cashier to the cash changer device.
  This method is triggered when a user logs on to Retail POS.
  
  **Parameters**
  1. *terminalID* – Terminal ID
  2. *operatorID* – Operator ID
  3. return value – Not used

• bool **Logout**()
  Logs out the current Retail POS operator from the cash changer device.
  This method is triggered when a user logs off Retail POS.
  
  **Parameters**
  1. return value – Not used

• CashChangerReturn **RegisterAmount**(decimal *amountDue*, string *receiptID*, ref decimal *amountRest*);
  Registers the amount from the cash changer to the transaction.
  This method is triggered from change-back operation, if a cash changer is enabled.
  
  **Parameters**
  1. *amountDue* – The amount that should be paid by the customer
  2. *receiptID* – The receipt ID of the transaction
  3. *amountRest* – Specifies what is left to pay if the amount that has been entered is not enough

• bool **Change**()
  Executes a change operation on the cash changer device.
  This method is triggered when a Change operation is selected from the Cash Change dialog box.

  **Parameters**
  1. return value – Not used

• bool **Reset**()
  Executes a reset operation in the cash changer device.
  This method is triggered when Retail POS must reset the device.

  **Parameters**
  1. return value – Not used

• bool **Exit**()
  Executes an exit operation on the cash changer.
  This method is triggered when Retail POS closes.

• bool **Regret**(CashChangerRegretType *regretType*)
  Executes regret operation in the cash changer.
This method is triggered when a sale is suspended, voided, or when a regret operation is selected from the **Cash Change** dialog box.

**Parameters**
- `regretType` – Type of regret to perform (All, One)

**bool GetAutoMode()**
Indicates whether the cash changer is in automode.

**Parameters**
- `return value` – True if in automode; otherwise, false.

**void SetAutoMode(bool mode)**
Sets the cash changer to automode.

**Parameters**
- `mode` – True if in automode; otherwise, false.

**void ConcludeTransaction(PosTransaction posTransaction)**
Handles the Conclude Sale operation.

This method is triggered when a sale is completed.

**Parameters**
- `posTransaction` – Transaction object

**ICashDrawer interface**
See "**Interfaces for peripherals**," later in this document.

**ICCTV interface**
The **ICCTV** interface provides methods to send output to a closed-circuit television (CCTV) system. You can display database information about transactions on the monitor while supervising Retail POS.

**Methods**
- **void CCTVErrorOutput(string text)**
  Sends the current error message in Retail POS to the CCTV.
  This method is triggered when Retail POS displays an error message.
  **Parameters**
  - `text` – Message text

- **void CCTVMessageOutput(string text)**
  Sends the current message in Retail POS to the CCTV.
  This method is triggered when Retail POS displays a message.
  **Parameters**
  - `text` – Message text
• void **CCTVOutput** (PosTransaction posTransaction, PosisOperations operationId, bool mainOperation, object operationInfo, string text)

Logs currently executing operation to the CCTV server.

This method is triggered post execution of every operation.

**Parameters**

- **posTransaction** – Transaction object that contains all the information about the transaction, such as items, payments, and so on
- **operationId** – The ID of the current operation
- **mainOperation** – Indicates whether the current operation is a main operation or support operation
- **operationInfo** – Extra information about the current operation
- **text** – Optional text supplied by Retail POS

**ICorporateCard interface**

The **ICorporateCard** interface provides methods to process and void card payments that are made with corporate cards. Corporate cards are customer cards or employee cards, and are issued and authorized by the company using Retail POS. These cards can be used as payments.

The card can be recognized as a corporate card by the Card service (see the preceding topic, "ICard interface"), which would then result in a call to the Corporate Card service. When the Corporate Card service processes are finished, the information can be sent back to the core.

**Methods**

- **Void ProcessCardPayment** (CardInfo cardInfo, decimal amount, object posTransaction)

  Processes the card payment and adds a payment line to the sale.

  This method is triggered by swiping a corporate card, manually entering the card information, or invoking the Pay Corporate Card operation.

  **Parameters**

  - **cardInfo** – The card info object that contains the detailed card information
  - **amount** – The amount used in the card transaction
  - **posTransaction** – All the information about the transaction

- **Void VoidCardPayment** (CardInfo cardInfo, object posTransaction)

  Voids a payment that has been added to a transaction.

  This method is called when the void payment operation is invoked.

  **Parameters**

  - **cardInfo** – The card info object containing the detailed card information
  - **posTransaction** - All the information about the transaction
Interfaces for Services and Triggers

ICreateDatabase interface
When no database exists for Retail POS, the ICreateDatabase interface creates a new database instance according to the Retail POS configuration file. This method can also add demo data if configured to do so, as specified in the configuration file.

Methods
- Void checkDatabase(string connectionString, string dataPhysicalFilePath)
  Installs or updates a Retail POS database and demo data.
  This method is triggered at application start.
  Parameters
  o connectionString – Database connection string
  o dataPhysicalFilePath – Physical location for the .mdb file
- Void CheckDefaultData(string connectionString, string dataAreaId)
  Currently not used.

Note
If you are customizing the CreateDatabase.dll, you must embed DatabaseScript.txt into the assembly as a resource, and the following files in the service project file must have Build Action set to Embedded Resource:
- DatabaseScript.txt
- DemoData\*. *
- DemoPics\*. *
- Upgrades\*. *

ICreditMemo interface
The ICreditMemo interface provides methods to handle store issued credit memos. Issuing a credit memo publishes and validates it at the head office through Retail Transaction Service. If the operation fails at any point, Retail POS removes the credit memo from the transaction and recalculates the transaction.

Methods
- void AuthorizeCreditMemoPayment (ref bool valid, ref string comment, ref string creditMemoId, ref string currencyCode, ref decimal amount, PosTransaction posTransaction)
  Validates a credit memo being used as a payment and reserves it at the head office.
  This method is triggered when a credit memo is used as payment.
  Parameters
  o valid – Returns true if the operation is successful; otherwise, false.
  o comment – A text message if the operation failed
  o creditMemoId – The ID of the credit memo
  o currencyCode – The currency code for the credit memo
  o amount – The amount to be used as payment
• **posTransaction** – All the information about the transaction

• **void GetCreditmemoBalance** (string creditMemoNumber, ref decimal balance)
  
  Returns the balance of a credit memo.
  
  This method is triggered by the "Credit memo balance" operation.

  **Parameters**
  
  o **giftCardId** – The ID of the credit memo
  
  o **balance** – The amount left on the credit memo

• **void IssueCreditMemo** (CreditMemoTenderLineItem creditMemoItem, RetailTransaction retailTransaction)
  
  Issues a gift certificate and adds it to the transaction.
  
  This method is triggered by the "Issue Gift Certificate" operation.

  **Parameters**
  
  o **creditMemoItem** – The payment line item
  
  o **retailTransaction** – All the information about the transaction

• **void UpdateCreditMemo** (string creditMemoNumber, decimal amount, RetailTransaction retailTransaction, CreditMemoTenderLineItem creditMemoTenderLineItem)
  
  Deducts the amount paid with the credit memo from the credit memo's balance at the head office.
  
  This method is triggered when a transaction that contains a credit memo payment is completed.

  **Parameters**
  
  o **creditMemoNumber** – The number of the credit memo
  
  o **amount** – The amount to be used as payment
  
  o **retailTransaction** – All the information about the transaction
  
  o **creditMemoTenderLineItem** - The payment line item

• **void VoidCreditMemoPayment** (ref bool voided, ref string comment, string creditMemoNumber, RetailTransaction retailTransaction)
  
  Releases the credit memo at the head office.
  
  This method is triggered when a credit memo payment line is voided.

  **Parameters**
  
  o **voided** – Returns true if the operation is successful; otherwise, false.
  
  o **comment** – A text message if the operation failed
  
  o **creditMemoNumber** – The number of the credit memo
  
  o **retailTransaction** – All the information about the transaction
ICurrency interface
The ICurrency interface provides various methods for currency, such as retrieving the local exchange rate, retrieving detailed information about specific currency types (including whether amounts are coins or bills), and converting one currency to another.

Methods
- void CurrencyToCurrency(string fromCurrencyCode, string toCurrencyCode, decimal orgValue)
  Converts an amount from one currency to another currency using the current exchange rate.
  
  Parameters
  - fromCurrencyCode – The currency of the orgValue
  - toCurrencyCode – Target currency for conversion
  - orgValue – The value to be converted
- CurrencyInfo DetailedCurrencyInfo(string currencyCode)
  Returns the CurrencyInfo object containing the current exchange rate and available denominations for the given currency.
  
  Parameters
  - currencyCode – Currency code; for example, USD
- decimal ExchangeRate(string currencyCode)
  Returns the latest exchange rate for a given currency code relative to default currency settings.
  
  Parameters
  - currencyCode – Currency code
- void GetAllCurrTypes(string localCurrencyCode, string usedCurrencyCode, ref ArrayList currTypes)
  Returns all available currency codes.
  
  Parameters
  - localCurrencyCode – The local currency code (which is excluded from list)
  - usedCurrencyCode – Not used
  - currTypes (out) – An array list containing currency codes

ICustomer interface
The ICustomer interface provides methods for interacting with customers. It provides UI to add customers, search for customers, and view information about customers.

Methods
- bool AddNew(ref Customer customer)
  Adds the new customer to the database.
  
  This method is triggered by the Customer Add operation.
Parameters

- Customer (out) – Newly added customer

- void AuthorizeCustomerAccountPayment(ref bool valid, ref string comment, ref string manualAuthenticationCode, string customerId, decimal amount, RetailTransaction transaction)

  Authenticates a payment from customer account.
  This method is triggered by the Pay Customer Account operation.

  Parameters
  
  - valid (out) – True if payment is valid; otherwise, false.
  - comment (out) – Error message if authentication failed
  - manualAuthenticationCode (out) – Authentication code for payment line
  - customerId – Customer ID for authentication
  - amount – Amount due for the sale
  - transaction – The Transaction object

- RetailTransaction Balance(RetailTransaction retailTransaction)
  
  Not used.

- Void BalanceReport()
  
  Prints a balance report for all customers.
  This method is triggered from the Customer Balance Report operation.

- Void Delete(string customerId)
  
  Not used.

- RetailTransaction EnterCustomerId(RetailTransaction retailTransaction)
  
  Not used.

- Bool Update(string customerId)
  
  Not used.

- PosTransaction Search(PosTransaction posTransaction)
  
  Searches for a customer and adds the customer to the transaction.
  This method is triggered by the Customer Search operation.

  Parameters
  
  - posTransaction – The posTransaction object for adding a customer to a transaction

- RetailTransaction Status(RetailTransaction retailTransaction)
  
  Not used.

- Void Transactions(string customerId)
  
  Shows transactions for the customer.
  This method is triggered by the Customer Transactions operation.

  Parameters
  
  - customerId – Customer ID
• Void **TransactionsReport**(string `customerId`)
  Prints customer transactions.
  This method is triggered by the Customer Transactions Report operation.
  **Parameters**
  o `customerId` – Customer ID

**IDialog interface**

The **IDialog** interface provides common implementations of various UI dialog boxes such as search, lookup, and various messages.

**Methods**

• void **ShowJournalDialog**(ref `JournalDialogResults` `dialogResult`, ref object `dialogResultObject`, ref `PosTransaction` `posTransaction`)
  Displays a dialog box listing journaled transactions.
  **Parameters**
  o `dialogResult` – The operation to be performed on the selected transaction
    Values:
    o Close = 0
    o PrintReceipt = 1
    o PrintInvoice = 2
    o ReturnTransaction = 3
    o TaxFree = 4
  o `dialogResultObject` – The string ID of the receipt that is selected
  o `posTransaction` – The current Retail POS transaction

• **DialogResult** **ShowMessage**(int `messageId`);
• **DialogResult** **ShowMessage**(int `messageId`, MessageBoxButtons `msgBoxButtons`);
• **DialogResult** **ShowMessage**(int `messageId`, MessageBoxButtons `msgBoxButtons`, MessageBoxIcon `msgBoxIcon`);
• **DialogResult** **ShowMessage**(string `message`, MessageBoxButtons `msgBoxButtons`, MessageBoxIcon `msgBoxIcon`)
  The **ShowMessage** methods display an arbitrary message, either by an ID or literal text, with optional combinations of buttons and icons. The DialogResult, MessageBoxButtons, and MessageBoxIcon enumerations are the standard .NET Framework System.Windows.Forms namespace types.

• **DialogResult** **ItemSearch**(int `howManyRows`, ref string `selectedItemID`)
  Invokes the **Item Search** dialog box.
  This method is invoked from **ButtonGrid** setup, Infocodes, and the default implementation of **IItem.ItemSearch**.
  This method returns DialogResult. OK indicates the successful selection of an item.
Interfaces for Services and Triggers

Parameters
- `howmanyRows` – The number of rows to display
- `selectedItemId` – The ID of the selected item

- **DialogResult GenericSearch(DataTable `dataTable`, ref DataRow `selectedRow`)**
  Displays a generic search dialog box.
  This method is invoked by Dimension selection.
  This method returns DialogResult. OK indicates the successful selection of an entry or row by the user.

  Parameters
  - `dataTable` – The data to display
  - `selectedRow` – The row selected in the given data table

- **DialogResult GenericLookup(DataTable `dataTable`, int `displayColumn`, ref DataRow `selectedRow`, decimal `sizeFactor`, string `defaultValue`)**
  Displays a generic lookup dialog box.
  This method is invoked from the **New Customer** dialog box.
  This method returns DialogResult. OK indicates the successful selection of an entry by the user.

  Parameters
  - `dataTable` – The table to display
  - `displayColumn` – The index of the column to be displayed
  - `selectedRow` – The row selected by the user
  - `sizeFactor` – The multiplier for the height and width of the dialog (Defaults to 1.0)
  - `defaultValue` – The default selected value for the lookup UI (for example, the value in the column indicated by the `displayColumn` parameter)

- **DialogResult PriceCheck(bool `useScanner`, PosTransaction `posTransaction`, ref string `inputText`)**
  Displays a dialog box for the price-check operation. The default implementation performs a price check, and then enables the selected item to be added to the current transaction.
  This method returns DialogResult. OK indicates that Retail POS should attempt to perform the Item Sale operation using a bar code value as output by the `inputText` parameter.

  Parameters
  - `useScanner` – Specifies whether to support a scanner (always called with a value of true)
  - `posTransaction` – A reference to the current transaction
  - `inputText` – Bar code of the selected item

- **DialogResult TenderDeclaration(ref PosTransaction `posTransaction`)**
  Displays UI to perform a tender declaration at the terminal.
  This method is invoked from Bank Drop, Safe Drop, and Tender Declaration operations.
  This method returns DialogResult. Retail POS continues with the operation and posts the transaction if the result is anything other than Cancel.
Parameters

- `posTransaction` – The current Retail POS transaction

**IDimension interface**

The **IDimension** interface retrieves the inventory dimensions attached to an item from the database.

**Methods**

- **void GetDimensionForVariant(ref Dimensions dimension)**
  
  Return the dimensions for a given variant ID.

  This method is triggered when Retail POS starts a return transaction.

  **Parameters**

  - `Dimension.VariantId` – variant ID
  - `dimension` (out) – `Dimension` object populated with data

- **DataTable GetDimensions(string itemId)**

  This method takes Item ID and returns a `DataTable` that contains available dimensions for the item.

  The method is triggered when a sale item with dimensions is added to a transaction.

  **Parameters**

  - `itemId` – Item ID added to the sale
  - `DataTable` (out) – Table that contains the following columns: VariantId, ColorId, Color, SizeId, Size, StyleId, Style, Config, ConfigId

**IDiscount interface**

The **IDiscount** interface provides methods for calculating discounts on items and transactions.

**Methods**

- **RetailTransaction CalculateDiscount(RetailTransaction retailTransaction)**

  Evaluates the entire transaction for any and all discounts that should apply. When triggered, all currently applied discounts are re-evaluated to grant the best possible offer to the customer.

  This method is triggered when items are added or removed from the transaction, quantities or units of measure are changed, or the customer is changed.

  This method returns the transaction with all discount modifications applied to the transaction and line items.

  **Parameters**

  - `retailTransaction` – The current transaction

- **void AddTotalDiscountAmount(RetailTransaction rt, decimal amountValue)**

  Applies the provided amount as a discount to the total transaction balance.

  This method is invoked by the Total Discount Amount operation.
Parameters
- $rt$ – The current retail transaction
- $amountValue$ – The currency amount of the discount

- **bool AuthorizeTotalDiscountAmount**(RetailTransaction $rt$, decimal $amountValue$, decimal $maxAmountValue$)
  Indicates whether the given discount amount is allowed.
  This method is invoked by the Total Discount Amount operation prior to calling AddTotalDiscountAmount.
  This method returns true if the discount amount is allowed and should continue; otherwise, false.

Parameters
- $rt$ – The current retail transaction
- $amountValue$ – The currency amount of the discount
- $maxAmountValue$ – The maximum allowed amount

- **void AddTotalDiscountPercent**(RetailTransaction $rt$, decimal $percentValue$)
  Applies the provided percentage value as a discount to the total transaction balance.
  This method is invoked by the Total Discount Percentage operation.

Parameters
- $rt$ – The current retail transaction
- $percentValue$ – The percentage value of the discount

- **bool AuthorizeTotalDiscountPercent**(RetailTransaction $rt$, decimal $percentValue$, decimal $maxPercentValue$)
  Indicates whether the given discount percentage is allowed.
  This method is invoked by the Total Discount Percentage operation prior to calling AddTotalDiscountPercentage.
  This method returns true if the discount percentage is allowed and should continue; otherwise, false.

Parameters
- $rt$ – The current retail transaction
- $percentValue$ – The percentage value of the discount
- $maxPercentValue$ – The maximum allowed percentage

- **void AddLineDiscountAmount**(SaleLineItem $lineltem$, LineDiscountItem $discountItem$)
  Applies a discount amount to a specific line item in a transaction.
  This method is invoked by the Line Discount Amount operation.

Parameters
- $lineltem$ – The current sales line
- $discountItem$ – The LineDiscountItem object to be applied to the sales line
- `bool AuthorizeLineDiscountAmount(SaleLineItem lineItem, LineDiscountItem discountItem, decimal maximumDiscountAmt)`
  Indicates whether the given discount amount is allowed.
  This method is invoked by the Line Discount Amount operation prior to calling AddLineDiscountAmount.
  This method returns true if the discount amount is allowed and should continue; otherwise, false.

  **Parameters**
  - `lineItem` – The current sales line
  - `discountItem` – The LineDiscountItem object to be applied to the sales line
  - `maximumDiscountAmt` – The maximum allowed amount

- `void AddLineDiscountPercent(SaleLineItem lineItem, LineDiscountItem discountItem)`
  Applies the percentage value to a specific line in a sale.
  This method is invoked by the Line Discount Percentage operation.

  **Parameters**
  - `lineItem` – The current sales line
  - `discountItem` – The LineDiscountItem object to be applied to the sales line

- `bool AuthorizeLineDiscountPercent(SaleLineItem lineItem, LineDiscountItem discountItem, decimal maximumDiscountPct)`
  Indicates whether the given discount percentage is allowed.
  This method is invoked by the Line Discount Percentage operation prior to calling AddLineDiscountPercentage.
  This method returns true if the discount amount is allowed and should continue; otherwise, false.

  **Parameters**
  - `lineItem` – The current sales line
  - `discountItem` – The LineDiscountItem object to be applied to the sales line
  - `maximumDiscountPct` – The maximum allowed percentage

**IDualDisplay interface**
See "Interfaces for peripherals," later in this document.

**IEFT interface**
The IEFT interface provides methods that can be used by external service providers to process card payments. Interface APIs provide the ability to authorize, refund, and void debit and credit card payments.
Methods

- `bool ProcessCardPayment(ref EFTInfo eftInfo, object posTransaction);`
  Processes the card payment by establishing a connection with the configured payment processor. After a connection to the broker is established, it attempts to authorize the card payment.
  This method returns true if the processing succeeded.
  This method is triggered from the card service when the card type is International Credit Card or International Debit Card.
  
  **Parameters**
  - `eftInfo` – A reference to an `EftInfo` object
  - `posTransaction` – The current transaction

- `bool VoidTransaction(ref EFTInfo eftInfo, object posTransaction);`
  Voids the card payment by establishing a connection with the configured payment processor. After a connection to the broker is established, it attempts to void the card payment.
  This method returns true if the processing succeeded.
  This method is triggered by the Void Payment operation.
  
  **Parameters**
  - `eftInfo` – A reference to an `EftInfo` object
  - `posTransaction` – The current transaction

- `void IdentifyCard(ref CardInfo cardInfo, ref EFTInfo eftInfo);`
  Identifies the card if a match with preconfigured card types is not found by Retail POS.
  This method is triggered by the Pay Card service and the Void Payment operation.
  
  **Parameters**
  - `cardInfo` – The card information
  - `eftInfo` – A reference to an `EftInfo` object

- `bool GetCardInfoAndAmount(ref CardInfo cardInfo);`
  Implements workflows supported by the card type. This method gets the card information and amount for the specific card.
  This method returns true if the system should continue with the authorization process; otherwise, false.
  This method is triggered by the Pay Card service.
  
  **Parameters**
  - `cardInfo` – The card information

- `void CapturePayment(EFTInfo eftInfo, object posTransaction);`
  Captures the card payment by establishing a connection with the configured payment processor. After a connection to the broker is established, it attempts to capture the authorized card payment.
  This method is triggered by the Pay Card service.
Parameters
- eftInfo – A reference to an EftInfo object
- posTransaction – The current transaction

**IEOD interface**
The IEOO interface provides End of Day, End of Shift, and X Report operations.

**Methods**
- **void CloseBatch(PosTransaction transaction)**
  This method performs a close batch operation. Closes the current open batch and prints a Z report.
  **Parameters**
  - transaction – The transaction object

- **void EndOfDay(EndOfDayTransaction transaction)**
  This method performs end-of-day operation and updates the transaction status.
  The method is triggered by the End of Day operation.
  **Parameters**
  - transaction – The transaction object

- **void EndOfShift(EndOfShiftTransaction transaction)**
  This method performs end-of-shift operation and updates the transaction status.
  The method is triggered by the End of Shift operation.
  **Parameters**
  - transaction – The transaction object

- **long GetCurrentBatchId()**
  This method returns the current batch ID that is associated with the transaction.

- **void PrintXReport(PosTransaction transaction)**
  This method prints the X report.
  The method is triggered by the Print X Report operation.
  **Parameters**
  - transaction – The transaction object

- **Void PrintZReport(PosTransaction transaction)**
  This method prints the Z report.
  The method is triggered by the Print Z Report operation.
  **Parameters**
  - transaction – The transaction object
IGiftCard interface
The IGiftCard interface provides methods to handle store-issued gift cards. By default, issuing a gift card publishes and validates it at the head office via Retail Transaction Service. If the operation fails at any point, Retail POS removes the gift card from the transaction and recalculates the transaction.

Implementation of these methods can be changed to integrate with a third-party gift card service.

Methods
- void AddToGiftCard (RetailTransaction retailTransaction, Tender gcTenderInfo)
  Add money to a gift card and adds it to the transaction.
  This method is triggered by the "Add to Gift Card" operation.
  Parameters
  o retailTransaction – All the information about the transaction
  o gcTenderInfo – The gift card tender information object
- void AuthorizeGiftCard (ref bool valid, ref string comment, ref string giftCardId, ref string currencyCode, ref decimal amount, PosTransaction posTransaction, Tender gcTenderInfo)
  Validates a gift card being used as a payment and reserves it at the head office.
  This method is triggered when a gift card is used as payment.
  Parameters
  o valid – Returns true if the operation is successful; otherwise, false
  o comment – A text message if the operation failed
  o giftCardId – The ID of the gift card
  o currencyCode – The currency code for the gift card
  o amount – The amount to be used as payment
  o posTransaction – All the information about the transaction
  o gcTenderInfo – The gift card tender information object
- void GiftCardBalance (PosTransaction posTransaction)
  Displays the balance amount of a gift card.
  This method is triggered by the "Gift Card Balance" operation.
  Parameters
  o posTransaction – All the information about the transaction
- void IssueGiftCard (PosTransaction posTransaction, Tender gcTenderInfo)
  Issues a gift card and adds it to the transaction.
  This method is triggered by the "Issue Gift Certificate" operation.
  Parameters
  o posTransaction – All the information about the transaction
  o gcTenderInfo – The gift card tender information object
• **void UpdateGiftCard** (ref bool updated, ref string comment, GiftCertificateTenderLineItem gcTenderLineItem)
  Deducts the amount paid with the gift certificate from the gift certificate's balance at the head office.
  This method is triggered when a sale that contains a gift certificate payment is completed.
  **Parameters**
  o **updated** - Returns true if the operation is successful; otherwise, false
  o **comment** – A text message if the operation failed
  o **gcTenderLineItem** – The gift card tender line item object

• **void VoidAddToGiftCard** (ref bool voided, ref string comment, GiftCertificateItem gcLineItem)
  Voids an addition to a gift card.
  This method is triggered when a gift card addition is voided or the transaction is voided.
  **Parameters**
  o **voided** - Returns true if the operation is successful; otherwise, false
  o **comment** – A text message if the operation failed
  o **gcLineItem** – The gift card line item object

• **void VoidGiftCard** (ref bool voided, ref string comment, GiftCertificateItem gcLineItem)
  Voids a gift card from a transaction.
  This method is triggered when a gift card line is voided or the transaction is voided.
  **Parameters**
  o **voided** - Returns true if the operation is successful; otherwise, false
  o **comment** – A text message if the operation failed
  o **gcLineItem** – The gift card line item object

• **void VoidGiftCardPayment** (ref bool voided, ref string comment, GiftCertificateTenderLineItem gcTenderLineItem)
  Releases the gift card at the head office.
  This method is triggered when a gift card payment line is voided.
  **Parameters**
  o **voided** - Returns true if the operation is successful; otherwise, false
  o **comment** – A text message if the operation failed
  o **gcTenderLineItem** – The gift card tender line item object

**IInfocodes interface**
The **IInfocodes** interface provides methods to handle infocode processing.
Methods

- **bool ProcessInfoCode** (PosTransaction posTransaction, string refRelation, string refRelation2, InfoCodeLineItem.TableRefId tableRefId, InfoCodeLineItem.InfoCodeType infoCodeType)

  This method processes the infocode based on its parameters.

  **Parameters**
  
  - *posTransaction* – The POS transaction object
  - *refRelation* – The first reference relation for the infocode
  - *refRelation2* – The second reference relation for the infocode
  - *tableRefId* – The table reference ID
  - *infoCodeType* – The infocode type

- **bool ProcessInfoCode** (PosTransaction posTransaction, decimal quantity, decimal amount, string refRelation, string refRelation2, string refRelation3, InfoCodeLineItem.TableRefId tableRefId, string linkedInfoCodeId, InfoCodeLineItem orgInfoCode, InfoCodeLineItem.InfoCodeType infoCodeType)

  This method processes the infocode based on its parameters.

  **Parameters**
  
  - *posTransaction* – The POS transaction object
  - *quantity* – The item quantity
  - *amount* – The item amount
  - *refRelation* – The first reference relation for the infocode
  - *refRelation2* – The second reference relation for the infocode
  - *refRelation3* – The third reference relation for the infocode
  - *tableRefId* – The table reference ID
  - *linkedInfoCodeId* – The linked infocode type
  - *orgInfoCode* – The infocode line item object
  - *infoCodeType* – The infocode type

- **void ProcessLinkedInfoCodes** (PosTransaction posTransaction, SaleLineItem saleLineItem, InfoCodeLineItem.TableRefId tableRefId, InfoCodeLineItem.InfoCodeType infoCodeType)

  This method processes the infocode based on its parameters.

  **Parameters**
  
  - *posTransaction* – The POS transaction object
  - *saleLineItem* – The sale line item object
  - *tableRefId* – The table reference ID
  - *infoCodeType* – The infocode type

- **void ProcessLinkedInfoCodes** (PosTransaction posTransaction, TenderLineItem tenderLineItem, string storeId, InfoCodeLineItem.TableRefId tableRefId, InfoCodeLineItem.InfoCodeType infoCodeType)

  This method processes the infocode based on its parameters.

  **Parameters**
- `posTransaction` – The POS transaction object
- `tenderLineItem` – The tender line item object
- `storeId` – The store ID
- `tableRefId` – The table reference ID
- `infoCodeType` – The infocode type

- void `ProcessLinkedInfoCodes`(`PosTransaction posTransaction, InfoCodeLineItem.TableRefId tableRefId, InfoCodeLineItem.InfocodeType infoCodeType`)

  This method processes the infocode based on its parameters.

  **Parameters**
  - `posTransaction` – The POS transaction object
  - `tableRefId` – The table reference ID
  - `infoCodeType` – The infocode type

**IItem interface**

The **IItem** interface is responsible for retrieving all item properties for a given line item and providing search UI for items.

**Methods**

- **SaleLineItem** `ProcessItem`(`SaleLineItem saleLineItem`)

  Gets and sets all properties on the item when an item is added to a transaction. Some properties may have been set already by Barcode Processing.

  This method is invoked by the Item Sale operation.

  This method returns a modified sale line item to be added to the transaction.

  **Parameters**
  - `saleLineItem` – The current line item

- bool `ItemSearch`(`ref string selectedItemId, int numberOfDisplayedRows`)

  Displays the item search UI.

  This method is invoked by the Item Search operation.

  This method returns true if the user chose an item (`selectedItemId` is the item chosen) and false if the user selected Cancel.

  **Parameters**
  - `selectedItemId` – The ID of the item selected in the UI
  - `numberOfDisplayedRows` – The number of rows to show in the UI

**IKKeyLock interface**

See "Interfaces for peripherals," later in this document.

**ILineDisplay interface**

See "Interfaces for peripherals," later in this document.
ILoyalty interface

The ILoyalty interface provides methods for the Loyalty business model such as adding a loyalty card to a transaction, calculating and adding loyalty points to a transaction, updating loyalty points (issued and used at the head office), and payments using loyalty cards.

Methods

- **bool AddLoyaltyCardToTransaction(CardInfo cardInfo, RetailTransaction retailTransaction);**
  Adds a scanned/entered loyalty card to the transaction. A new LoyaltyItem is instantiated, the properties are set, and the LoyaltyItem is assigned to the current RetailTransaction.
  If a previous loyalty item exists on the transaction, the system should prompt the user to overwrite the existing loyalty item or cancel the operation.
  This method is triggered when a swiped card is identified as a loyalty card.

  **Parameters**
  - *cardInfo* – Detailed information about the loyalty card
  - *retailTransaction* – The entire sales transaction

- **void AddLoyaltyItem(CardInfo cardInfo, RetailTransaction retailTransaction);**
  Reserved for future use.

- **void AddLoyaltyPoints(RetailTransaction retailTransaction);**
  If there is a loyalty record in the sales transaction, this method calculates loyalty points and adds them to the transaction. After this method has been called, the retailTransaction has to be updated with the latest status.
  This method is triggered when the sales transaction is concluded.

  **Parameters**
  - *retailTransaction* - The entire sales transaction

- **void AddLoyaltyPayment(CardInfo cardInfo, decimal amount, RetailTransaction retailTransaction);**
  When a customer pays with a loyalty card, the payment transaction is added to the sale.
  A new LoyaltyTenderLineItem is instantiated, the properties are set and the LoyaltyTenderLineItem is assigned to retailTransaction.
  This method is triggered when a loyalty card is used as payment method in a sale.

  **Parameters**
  - *cardInfo* – Detailed information about the loyalty card
  - *amount* – The amount that is being paid for with the loyalty card
  - *retailTransaction* – The entire sales transaction

- **void VoidLoyaltyPayment(CardInfo cardInfo, RetailTransaction retailTransaction);**
  Reserved for future use.

- **void UpdateIssuedLoyaltyPoints(RetailTransaction retailTransaction, LoyaltyItem loyaltyItem);**
  Updates the issued loyalty points for the sale being concluded. Delegates task to the Transaction service.
  This method is triggered by the ConcludeTransaction operation.
Parameters
- *retailTransaction* – The entire sales transaction
- *loyaltyItem* – Contains all loyalty information

- `void UpdateUsedLoyaltyPoints(RetailTransaction retailTransaction, LoyaltyTenderLineItem loyaltyTenderItem);`
  Updates the used loyalty points for the sale being concluded. Delegates task to the Transaction service.
  This method is triggered by the ConcludeTransaction operation.
  Parameters
  - *retailTransaction* - The entire sales transaction
  - *loyaltyTenderItem* – The loyalty payment line

**IMSR interface**
See "Interfaces for peripherals," later in this document.

**IPeripherals interface**
See "Interfaces for peripherals," later in this document.

**IPinPad interface**
See "Interfaces for peripherals," later in this document.

**IPrice interface**
The *IPrice* interface provides methods for getting prices for items.

Methods
- RetailTransaction *GetPrice*(RetailTransaction *retailTransaction*)
  Gets the price for an item when it is added to the transaction or when quantity changes.
  This method returns a new (updated) copy of the retail transaction (not using a reference value) where item prices have been set.
  Parameters
  - *retailTransaction* – The current retail transaction

- RetailTransaction *UpdateAllPrices*(RetailTransaction *retailTransaction*, bool *restoreItemPrices*);
  Updates all prices in a transaction when a customer is added, modified, or deleted, and when the unit of measure changes for a line item.
  This method returns a new (updated) copy of the retail transaction (not using a reference value) where item prices have been set or updated.
  Parameters
o  **retailTransaction** – The current retail transaction
o  **restoreItemPrices** – If this value is true, prices that have been overridden return to the original price

- **decimal GetBasicPrice(string itemId);**
  Returns the base price of an item.

  **Parameters**
  o  **itemId** – The unique item ID as stored in the inventory table

- **decimal GetItemPrice(string itemId, string unitOfMeasure);**
  Determines whether the item is a part of a price agreement, and if so, returns that price.
  This method is triggered when the **Item Search** dialog is displayed.

  **Parameters**
  o  **itemId** – The ID of the item to be checked
  o  **unitOfMeasure** – The unit of measure to be checked

- **bool ItemInPromotion(string itemId, string barcode);**
  This method determines whether the item is in a promotion. This method is currently not called from anywhere in the Retail POS code.
  Returns true if at least one promotion item exists for the item.

  **Parameters**
  o  **itemId** – ID of the item
  o  **barcode** – The optional bar code of the item

**IPrinter interface**
See "**Interfaces for peripherals,**" later in this document.

**IPrinting interface**
The **IPrinting** interface provides methods for previewing and printing receipts.

**Methods**
- **bool PrintReceipt(FormType formType, PosTransaction posTransaction, bool copyReceipt)**
  Prints the receipt for the transaction.
  The method is triggered when a transaction (sale, customer payment, sales order, or sales invoice transaction) is completed.

  **Parameters**
  o  **formType** – FormType.Receipt
  o  **posTransaction** – The **transaction** object containing all sales and payment line items
  o  **copyReceipt** – True if reprinting the receipt; otherwise, false.

- **void PrintCardReceipt(FormType formType, PosTransaction posTransaction, TenderLineItem tenderLineItem, bool copyReceipt)**
Prints the card payment receipt.
The method is triggered when transaction containing card payment lines is concluded. The method is called for each card payment line item in a transaction.

Parameters

- `formType` – FormType.CardReceiptForShop or FormType.CardReceiptForCust or FormType.CardReceiptForShopReturn or FormType.CardReceiptForCustReturn
- `posTransaction` – The transaction object containing all sales, payment line items
- `tenderLineItems` – The tender line-item collection
- `copyReceipt` – True if reprinting the receipt; otherwise, false.

- **void** PrintCardReceipt(FormType formType, PosTransaction posTransaction, EFTInterface.EFTInfo eftInfo, bool copyReceipt)
  Not used.

- **void** PrintCustomerReceipt(FormType formType, PosTransaction posTransaction, TenderLineItem tenderLineItem, bool copyReceipt)
  Prints the card payment receipt.
  This method is triggered when the transaction that contains customer payment lines is concluded. The method is called for each customer-payment line item in a transaction.

  Parameters

  - `formType` – FormType.CustAcntReceiptForShop or FormType.CustAcntReceiptForCust or FormType.CustAcntReceiptForShopReturn or FormType.CustAcntReceiptForCustReturn
  - `posTransaction` – The transaction object containing all sales and payment line items
  - `tenderLineItems` – The tender line-item collection
  - `copyReceipt` – True if reprinting the receipt; otherwise, false.

- **void** PrintFloatEntryReceipt(PosTransaction posTransaction)
  Prints a float-entry receipt.
  This method is triggered when a float-entry transaction is made.

  Parameters

  - `posTransaction` – FloatEntryTransaction object

- **void** PrintRemoveTenderReceipt(PosTransaction posTransaction)
  Prints a tender-removal receipt.
  Triggered when a Tender removal transaction is made.

  Parameters

  - `posTransaction` – RemoveTenderTransaction object

- **void** PrintCreditMemo(FormType formType, PosTransaction posTransaction, TenderLineItem tenderLineItem, bool copyReceipt)
  Prints the credit memo receipt.
  This method is triggered when a transaction that contains credit-memo tender lines is concluded.

  Parameters
• formType – FormType.CreditMemo
• posTransaction – The CreditMemoTenderLineItem object
• tenderLineItems – The tender-lines item collection
• copyReceipt – True if reprinting the receipt; otherwise, false.

• void PrintCreditMemoBalance(FormType formType, Decimal balance, bool copyReceipt)
  Not used.

• bool PrintInvoice(PosTransaction posTransaction, bool copyInvoice, bool printPreview)
  Prints the invoice receipt.
  This method is triggered when the Print Invoice button (in the Journal dialog box) is clicked.
  Parameters
  o posTransaction – The transaction object
  o copyInvoice – True if reprinting the receipt; otherwise, false.
  o printPreview – Not used

• bool ShowPrintPreview(FormType formType, PosTransaction posTransaction)
  Displays the receipt preview dialog.
  This method is triggered before printing a receipt if a preview is requested.
  Parameters
  o formType – The FormType object
  o posTransaction – The transaction object

• void PrintTenderDeclaration(PosTransaction posTransaction)
  Prints a tender-declaration receipt.
  This method is triggered when a tender declaration transaction is made.
  Parameters
  o posTransaction – The tenderDeclarationTransaction object

• void PrintBankDrop(PosTransaction posTransaction)
  Prints a bank drop receipt.
  This method is triggered when a bank drop transaction is made.
  Parameters
  o posTransaction – The BankDropTransaction object

• void PrintSafeDrop(PosTransaction posTransaction)
  Prints a safe drop receipt.
  This method is triggered when a safe-drop transaction is made.
  Parameters
  o posTransaction – SafeDropTransaction object
• void PrintGiftCertificate(FormType formType, PosTransaction posTransaction, GiftCertificateItem giftCertificateItem, bool copyReceipt)

Prints the gift card balance receipt. This method is triggered when a transaction that contains gift card line items is concluded.

Parameters
- formType – FormType.GiftCertificate
- posTransaction – The transaction object
- giftCertificateItem – The GiftCertificateItem object
- copyReceipt – True if reprinting the receipt; otherwise, false.

IPurchaseOrder interface
The IPurchaseOrder interface controls how Retail POS displays purchase orders when the cashier invokes the Receive Purchase Orders operation.

Methods
• void ShowPurchaseOrderList ()

Displays a list of purchase orders that can be received. This method is triggered by the ShowPurchaseOrder operation.

• void ReceivePurchaseOrder (string receiptNumber)

Displays a specific purchase order to be received. This method is triggered only from the purchase order list in Retail POS.

Parameters
- receiptNumber – The receipt number of the purchase order to be received

IRFID interface
The IRFID interface provides methods that process RFID scanned items.

Methods
• ScanInfo[] GetUnProcessedRFIDs()

Returns newly scanned items by RFID device that Retail POS must add to the sale. This method is triggered when a scan-complete event is received from an RFID device.

• Void MarkProcessedRFIDs(RetailTransaction transaction)

Associates a transaction with scanned items that are sent to Retail POS in the GetUnProcessedRFID call. This method is triggered when all RFID scanned items have been added to a transaction.

Parameters
- transaction – The transaction object

• Void ConcludeRFIDs()

Not used.
IRFIDScanner interface
See "Interfaces for peripherals," later in this document.

IRounding interface
The IRounding interface contains various utility methods used for rounding decimal values for display, printing, and rounding based on currency rules. This interface has many overloaded methods that are all self-explanatory and are not listed here.

ISalesInvoice interface
The ISalesInvoice interface has a default implementation for handling sales-invoice line items in the Retail POS transaction. The default implementation displays UI that enable the user to select from available sales invoices for the current customer and perform various Retail POS tasks with them.

Method
- void SalesInvoices(ref PosTransaction posTransaction)
  Manages sales-invoice line items.
  This method is triggered when the Sale Invoice operation is called.
  Parameters
  o posTransaction – The transaction object

ISalesOrder interface
The ISalesOrder interface has a default implementation for managing sales-order line items in the Retail POS transaction. The default implementation displays UI that enable the user to select from available sales orders for the current customer and perform various Retail POS tasks with them, such as applying a payment to the order, creating pick/packing lists, and so on.

Method
- void SalesOrders(ref PosTransaction posTransaction);
  Manages sales order items.
  This method is triggered when the Sales Order operation is called.
  Parameters
  o posTransaction – The transaction object
- void PriceOverride(ref PosTransaction posTransaction, object operationInfo);
  Manages the price-override operation for sales order.
  This method is triggered when the Price Override operation is called on a sales-order line item.
  Parameters
  o posTransaction – The transaction object
  o operationInfo – Additional information for the operation
**IScale interface**
See "Interfaces for peripherals," later in this document.

**IScanner interface**
See "Interfaces for peripherals," later in this document.

**IStockCount interface**
The IStockCount interface controls how Retail POS displays the Stock Count operation.

**Methods**
- void PerformStockCount ()
  Displays the user interface in Retail POS for creating, editing, and submitting stock counts.
  This method is triggered by the Stock Count operation.

**IStoreInventoryServices interface**
The IStoreInventoryServices interface controls how Retail POS retrieves and commits store inventory information to and from Microsoft Dynamics AX.

**Methods**
- void CommitStockCounts ()
  Commits all pending (uncommitted) stock count entries in the local database. The default implementation uses Retail Transaction Service to communicate with Microsoft Dynamics AX.
- void GetPurchaseOrderReceipts ()
  Retrieves a list of all new purchase order receipt records (header and line details) from the head office. The default implementation uses Retail Transaction Service to retrieve the information from Microsoft Dynamics AX.
- void CommitPurchaseOrderReceipt (string receiptID)
  Commits a pending purchase order receipt to the head office. The default implementation uses Retail Transaction Service to send the information to Microsoft Dynamics AX.
  **Parameters**
  - receiptID – The string ID of the purchase order receipt to be committed

**ITax interface**
The ITax interface provides methods for retrieving and calculating taxes for the transaction.

**Methods**
- RetailTransaction CalculateTax(RetailTransaction retailTransaction)
  Calculates the taxes for the current transaction.
  This method is triggered when items are added to the transaction, prices are changed, quantities are changed, or discounts are applied.
Returns a new copy of the given parameter \textit{retailTransaction}.

**Parameters**
- \textit{retailTransaction} – Information about the current transaction

- \textbf{RetailTransaction \texttt{RecalcAllTax}(RetailTransaction \texttt{retailTransaction});}

\begin{itemize}
\item \textbf{Note}
\end{itemize}

This method will be removed from future releases of Microsoft Dynamics AX for Retail.

Recalculates the tax on the given transaction exactly in the same manner as in the RetailTransaction \textbf{Calculate Tax} method.

This method is triggered only when a customer is added, changed, or cleared from the transaction.

Returns a new copy of the given parameter \textit{retailTransaction}.

**Parameters**
- \textit{retailTransaction} – Information about the current transaction

- \textbf{DialogResult \texttt{TaxOverrideList}(TaxOverrideBy \texttt{overrideType}, out string \texttt{selectedTaxOverride});}

Displays a list of available sales tax overrides.

**Parameters**
- \texttt{overrideType} – The type of sales tax override
- \texttt{selectedTaxOverride} – The selected sales tax override

- \textbf{void \texttt{CalculateTax}(BaseSaleItem \texttt{lineItem}, RetailTransaction \texttt{retailTransaction});}

Performs a simple tax check for an item. The function calculates the tax amount and may reset the Price variable for tax inclusive cases.

**Parameters**
- \texttt{lineItem} – The line item to be calculated
- \texttt{retailTransaction} – Information about the current transaction

- \textbf{void \texttt{CalcTaxIncluded}(SaleLineItem \texttt{lineItem}, RetailTransaction \texttt{rt});}

Performs a simple tax check for an item. The function calculates the tax amount and sets the \texttt{PriceInclTax} variable.

This method is not called from Retail POS directly, but used by the \textbf{CalculateTax} and \textbf{RecalculateTax} methods.

**Parameters**
- \texttt{lineItem} – The sales line to be calculated
- \texttt{rt} – The current retail transaction

- \textbf{void \texttt{CalcTaxExcluded}(SaleLineItem \texttt{lineItem}, RetailTransaction \texttt{rt});}

Performs a simple tax check for an item. The function calculates the tax amount and sets the \texttt{PriceExclTax} variable.

This method is not called from Retail POS directly, but used by the \textbf{CalculateTax} and \textbf{RecalculateTax} methods.
Parameters

- `lineItem` – The sales line to be calculated
- `rt` – The current retail transaction

### ITaxProvider interface

The ITax interface calls a collection of ITaxProvider instances. The default ITaxProvider computes taxes based on the existing Microsoft Dynamics AX tax configuration from the local database. However, additional ITaxProvider instances can be utilized to support additional tax scenarios without modifying the default provider. Each ITaxProvider instance is given the entire RetailTransaction object, and is responsible for computing tax results for all line items and appending them to a Tax collection (per line item).

To utilize an additional ITaxProvider instance, you must modify the Tax.dll.config file. This file specifies which TaxProviders Retail POS should load and call when taxes need to be calculated. Retail POS loads the .dlls mentioned in the "ProviderAssemblies" section (green in the sample below), and calls the classes mentioned in the "ProviderClassnames" section (yellow in the sample below). Tax providers are called in the order listed. Note that the special character "**" is used to indicate the default, built-in Retail POS tax provider.

The sample below shows that the MyProvider.dll will be loaded, and the MyNamespace.MyTaxCodeProvider class (which should implement the ITaxProvider interface – defined in the TaxiInterface assembly) will be called after the default tax engine.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
      <section name="Tax.Properties.Settings" type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" allowExeDefinition="MachineToLocalStorage" requirePermission="false"/>
    </sectionGroup>
  </configSections>
  <userSettings>
    <Tax.Properties.Settings>
      <setting name="ProviderAssemblies" serializeAs="Xml">
        <value>
            <string>MyProvider.dll</string>
          </ArrayOfString>
        </value>
      </setting>
      <setting name="ProviderClassNames" serializeAs="Xml">
        <value>
            <string>MyNamespace.MyTaxCodeProvider</string>
          </ArrayOfString>
        </value>
      </setting>
    </Tax.Properties.Settings>
  </userSettings>
</configuration>
```
ITenderRestriction interface

The ITenderRestriction interface has implementations for indicating whether any item in the transaction is restricted from being paid for with the currently added payment method.

Retail POS calculates a payable amount composed of all the items for which this tender is a valid payment. If the payable amount does not match the transaction total (that is, there are items that are excluded from the tender), the cashier is prompted with a list of the items and can choose to continue (which clears the restriction) or cancel.

Methods

- decimal **FindTenderRestriction**(RetailTransaction *retailTransaction*, CardInfo *cardInfo*)
  
  Returns the amount that is allowed to be paid with the corporate card.
  
  This method is triggered when corporate card payment line is being added to transaction.

  **Parameters**
  
  - *retailTransaction* – The transaction object
  - *cardInfo* – The corporate card information object

- void **ClearTenderRestriction**(RetailTransaction *retailTransaction*)
  
  Removes restrictions from the payment method in the transaction.
  
  This method is triggered when corporate-card processing is canceled.

  **Parameters**
  
  - *retailTransaction* – The transaction object
Interfaces for peripherals

ICashDrawer interface
This interface includes methods for a cash drawer.

Methods
• void Load()
  This method is called at application startup and loads/configures the cash drawer, if enabled.
• void Unload()
  This method is called when the application is closed. The cash drawer is released in this method.
• void OpenDrawer()
  This method is called to open the cash drawer.
• bool CapStatus()
  This method is called to get the ability of the cash drawer to report its status, opened or closed.
• bool DrawerOpen()
  This method is called to get the current status of the cash drawer, whether it is open or not.

IDualDisplay interface
This interface includes methods for a dual display that show sales details, advertisements, or web pages on a second monitor attached to the Retail POS terminal.

Methods
• void Load()
  This method is called at application startup and loads/configures the dual display, if enabled.
• void Unload()
  This method is called when the application is closed. The dual display is released in this method.
• void ShowTransaction(PosTransaction posTransaction)
  This method is called to update the transaction on the dual display.
  Parameters
  • posTransaction – The POS transaction object

IKeyLock interface
This interface includes methods for a key lock.
Methods

- void Load()
  This method is called at application startup and loads/configures the key lock, if enabled.
- void Unload()
  This method is called when the application is closed. The key lock is released in this method.
- bool SupervisorPosition()
  This method is called to check if the key lock is in the supervisor position.
- bool LockedPosition()
  This method is called to check if the key lock is in the locked position.

ILineDisplay interface

This interface includes methods for a line display.

Methods

- void Load()
  This method is called at application startup and loads/configures the line display, if enabled.
- void Unload()
  This method is called when the application is closed. The line display is released in this method.
- void ClearText()
  This method is called to clear all text from the line display.
- void DisplayText(string text)
  This method is called to display a single line of text on the line display.

  Parameters
  - text – The text to display

- void DisplayText(string line1Text, string line2Text)
  This method is called to display a double line of text on the line display.

  Parameters
  - line1Text – The text to display on line 1
  - line2Text – The text to display on line 2

- void DisplayItem(SaleLineItem saleLineItem)
  This method is called to display a sale line item on the line display.

  Parameters
  - saleLineItem – The sale line item object

- void DisplayTotal(string amount)
  This method is called to display the total sales amount on the line display.

  Parameters
  - amount – The total amount
• void **DisplayBalance**(string *amount*)
  This method is called to display the balance due on the line display.

  **Parameters**
  o *amount* – The balance due amount

**IMSR interface**
This interface includes methods for a magnetic stripe reader (MSR).

**Methods**
• void **Load**()
  This method is called at application startup and loads/configures the MSR, if enabled.

• void **Unload**()
  This method is called when the application is closed. The MSR is released in this method.

• void **EnableForSwipe**()
  This method is called to enable swiping on the MSR.

• void **DisableForSwipe**()
  This method is called to disable swiping on the MSR.

**IPeripherals interface**
This interface exposes all device specific class instances.

**Methods**
• void **Load**()
  This method is called at application startup and loads/configures all enabled peripherals.

• void **Unload**()
  This method is called when the application is closed. All enabled peripherals are released in this method.

**IPinPad interface**
This interface includes methods for a PIN pad.

**Methods**
• void **Load**()
  This method is called at application startup and loads/configures the PIN pad, if enabled.

• void **Unload**()
  This method is called when the application is closed. The PIN pad is released in this method.

• void **BeginTransaction**(decimal *amount*, string *accountNumber*)
  This method is called to begin a transaction on the PIN pad.
Parameters
- amount – The amount to display
- accountNumber – The number to display

- void EndTransaction(bool normal)
  This method is called to end a transaction on the PIN pad.

Parameters
- normal – The normal or abnormal end of the transaction

IPrinter interface
This interface includes methods for printers.

Methods
- void Load()
  This method is called at application startup and loads/configures the printer, if enabled.
- void Unload()
  This method is called when the application is closed. The printer is released in this method.
- void PrintReceipt(string text)
  This method is called to print a receipt.

Parameters
- text – The date to print in text format

- void PrintSlip(string text)
  This method is called to print a receipt as a slip.

Parameters
- text – The date to print in text format

- void PrintSlip(string header, string details, string footer)
  This method is called to print a receipt as a slip.

Parameters
- header – The header part of the slip
- details – The details part of the slip
- footer – The footer part of the slip

- void WindowsPrinting(string textToPrint, string printerName)
  This method is called to print a receipt on a Windows® printer, not an OPOS printer.

Parameters
- textToPrint – The text to print
- printerName – The name of the Windows printer

IRFIDScanner interface
This interface includes methods for a radio frequency identification (RFID) scanner.
Methods
- **void Load()**
  This method is called at application startup and loads/configures the RFID scanner, if enabled.
- **void Unload()**
  This method is called when the application is closed. The RFID scanner is released in this method.
- **void DisableForScan()**
  This method is called to disable scanning.
- **void ReEnableForScan()**
  This method is called to enable scanning.
- **void ConcludeRFID()**
  This method is called to conclude the pending RFID scanner.

**IScale interface**
This interface includes methods for a scale.

Methods
- **void Load()**
  This method is called at application startup and loads/configures the scale, if enabled.
- **void Unload()**
  This method is called when the application is closed. The scale is released in this method.
- **void ReadFromScale()**
  This method is called to read the weight from the scale.

**IScanner interface**
This interface includes methods for a bar code scanner.

Methods
- **void Load()**
  This method is called at application startup and loads/configures the scanner, if enabled.
- **void Unload()**
  This method is called when the application is closed. The scanner is released in this method.
- **void DisableForScan()**
  This method is called to disable scanning.
- **void ReEnableForScan()**
  This method is called to enable scanning.
Interfaces for triggers

IApplicationTriggers interface

The IApplicationTriggers interface contains triggers for application startup and logon operations.

Triggers

- **Void ApplicationStart()**
  Triggers once, when the application starts.

- **Void ApplicationStop()**
  Triggers once, just before the application stops.

- **Void LoginWindowVisible()**
  Triggers when the logon window gets visible.

- **void PreLogon(ref PreTriggerResults results, string operatorId, string name)**
  Triggers just before the login operation is about to be run. This operation can be stopped by setting PreTriggerResults.RunOperation to false.

  **Parameters**
  
  - `results (out)` – PreTriggerResults
  - `operatorId` – Operator ID
  - `name` – Operator name

- **void PostLogon(bool loginSuccessful, string operatorId, string name)**
  Triggers after the login operation has been executed.

  **Parameters**
  
  - `loginSuccessful` – True if logon was successful; otherwise, false.
  - `operatorId` – Operator ID
  - `name` – Operator name

- **void Logoff(string operatorId, string name)**
  Triggers when logoff operation is run.

  **Parameters**
  
  - `operatorId` – Operator ID
  - `name` – Operator name
ICashManagementTriggers interface
The ICashManagementTriggers interface contains triggers for cash management operations.

Triggers
- void PreTenderDeclaration(ref PreTriggerResults results, ref PosTransaction transaction)
  Triggers before the tender-declaration operation begins.
  Parameters
    - results – PreTriggerResults
    - transaction – The PosTransaction object
- void PostTenderDeclaration(ref PosTransaction transaction)
  Triggers after the tender-declaration operation is executed.
  Parameters
    - transaction – The PosTransaction object

ICustomerTriggers interface
The ICustomerTriggers interface contains triggers for customer operations.

Triggers
- void PreCustomerClear(ref PreTriggerResults results, ref PosTransaction posTransaction)
  Triggers before clearing a customer from the transaction.
  Parameters
    - results – PreTriggerResults
    - posTransaction – The PosTransaction object
- void PostCustomerClear(ref PosTransaction posTransaction)
  Triggers after clearing a customer from the transaction.
  Parameters
    - posTransaction – The PosTransaction object

IDiscountTriggers interface
The IDiscountTriggers interface contains triggers for discount operations.

Triggers
- void PreLineDiscountAmount(ref PreTriggerResults results, ref PosTransaction posTransaction, int Lineld)
  Triggers before adding the line discount amount.
  Parameters
    - results – PreTriggerResults
    - posTransaction – The PosTransaction object
o  `lineId` – The sale line-item number in the transaction

- void `PreLineDiscountPercent` (ref `PreTriggerResults results`, ref `PosTransaction posTransaction`, int `lineId`)
  Triggers before adding line-discount percent.
  Parameters
  o  `results` – `PreTriggerResults`
  o  `posTransaction` – The `PosTransaction` object
  o  `lineId` – The sale line-item number in the transaction

**IInfocodeTriggers interface**
The `IInfocodeTriggers` interface contains triggers for infocode operations.

Triggers
- void `PreProcessInfocode` (ref `PreTriggerResults results`, ref `PosTransaction posTransaction`, InfoCodeLineItem `TableRefId tableRefId`)
  Triggers before the infocode is being processed.
  Parameters
  o  `results` (out) - `PreTriggerResults`
  o  `posTransaction` – The `PosTransaction` object
  o  `tableRefId` – InfoCode Table Reference ID

- void `PostProcessInfocode` (ref `PreTriggerResults results`, ref `PosTransaction posTransaction`, InfoCodeLineItem `TableRefId tableRefId`)
  Triggers after the infocode has been processed.
  Parameters
  o  `results` - Not used
  o  `posTransaction` – The `PosTransaction` object
  o  `tableRefId` – InfoCode Table Reference ID

**IItemTriggers interface**
The `IItemTriggers` interface contains triggers for item operations.

Triggers
- void `PreSale` (ref `PreTriggerResults results`, ref `SaleLineItem saleLineItem`, ref `PosTransaction posTransaction`)
  Triggers before adding the item to the transaction but after all item properties have been retrieved from the database. Note that the item’s dimensions have not been checked.
  Parameters
  o  `results` (out) - `PreTriggerResults`
  o  `saleLineItem` – The sale line item that is added
Interfaces for Services and Triggers

Interfaces for triggers

- **posTransaction** – The PosTransaction object

  - void **PostSale**(ref PosTransaction posTransaction)
    Triggers after adding the item to the transaction. Prices and discounts have been calculated, but the event is triggered before processing any infocodes or linked items.
    
    Parameters
    - posTransaction – The PosTransaction object

  - void **PreReturnItem**(ref PreTriggerResults results, ref PosTransaction posTransaction)
    Triggers before adding a return item to the transaction. The return operation only sets the register to a return state. It is the item-sale operation that really returns the item (that is, sells it with a negative quantity). Therefore, programming the return triggers only affects whether the register can enter the return state.
    
    Parameters
    - results (out) - PreTriggerResults
    - posTransaction – The PosTransaction object

  - void **PostReturnItem**(ref PosTransaction posTransaction)
    Not used.

  - void **PreVoidItem**(ref PreTriggerResults results, ref PosTransaction posTransaction, int lineId)
    Triggers before voiding a line item.
    
    Parameters
    - results (out) - PreTriggerResults
    - posTransaction – The PosTransaction object
    - lineId – The sale line-item number in the transaction

  - void **PostVoidItem**(ref PosTransaction posTransaction, int lineId)
    Triggers after voiding a line item.
    
    Parameters
    - posTransaction – The PosTransaction object
    - lineId – Sale line-item number in transaction

  - void **PreSetQty**(ref PreTriggerResults results, ref SaleLineItem saleLineItem, ref PosTransaction posTransaction, int lineId)
    Triggers before overriding the quantity of a line item.
    
    Parameters
    - results (out) - PreTriggerResults
    - saleLineItem – The sale line item
    - posTransaction – The PosTransaction object
    - lineId – The sale line-item number in the transaction
• void **PostSetQty**(ref PosTransaction *posTransaction*, ref SaleLineItem *saleLineItem*)

  Triggers after overriding the quantity of a line item.

  **Parameters**
  o  *posTransaction* – The **PosTransaction** object
  o  *saleLineItem* – Sale line item

• void **PrePriceOverride**(ref PreTriggerResults *results*, ref SaleLineItem *saleLineItem*, ref PosTransaction *posTransaction*, int *linId*)

  Triggers before overriding the price of a line item.

  **Parameters**
  o  *results* (out) - PreTriggerResults
  o  *saleLineItem* – The sale line item
  o  *posTransaction* – The **PosTransaction** object
  o  *linId* – The sale line-item number in the transaction

• void **PostPriceOverride**(ref PosTransaction *posTransaction*, ref SaleLineItem *saleLineItem*)

  Triggers after overriding the price of a line item.

  **Parameters**
  o  *posTransaction* – The **PosTransaction** object
  o  *saleLineItem* – The sale line item

**IPaymentTriggers interface**

The **IPaymentTriggers** interface contains triggers for payment operations.

**Triggers**

• void **PrePayCustomerAccount**(ref PreTriggerResults *results*, ref PosTransaction *posTransaction*, decimal *amount*)

  Triggers before a customer account payment is made.

  **Parameters**
  o  *results* (out) - PreTriggerResults
  o  *posTransaction* – The **PosTransaction** object
  o  *amount* – The amount that is paid

• void **PrePayCardAuthorization**(ref PreTriggerResults *results*, ref PosTransaction *posTransaction*, ref CardInfo *cardInfo*, ref decimal *amount*)

  Triggers before a card payment is made.

  **Parameters**
  o  *results* (out) - PreTriggerResults
  o  *posTransaction* – The **PosTransaction** object
  o  *cardInfo* – Detailed information about card
  o  *amount* – The amount that is paid
• **void** **PrePayment**(ref **PreTriggerResults** **results**, ref **PosTransaction** **posTransaction**, object **posOperation**, string **tenderId**)

Triggers on each payment operation, before setting an amount, swiping any cards, entering any input, or showing any dialog.

**Parameters**
- **results** (out) - **PreTriggerResults**
- **posTransaction** – The **PosTransaction** object
- **posOperation** – The **operation** object
- **tenderId** – The ID of the tender type

• **void** **OnPayment**(**RetailTransaction** **retailTransaction**)

Triggers when the payment has been added to the transaction.

**Parameters**
- **retailTransaction** – The **RetailTransaction** object

**ISuspendTriggers** interface

The **ISuspendTriggers** interface contains triggers for suspending a transaction and recalling transaction operations.

**Triggers**
• **void** **PreSuspendTransaction**(ref **PreTriggerResults** **results**, ref **PosTransaction** **posTransaction**)

Triggers before the suspended transaction operation executes.

**Parameters**
- **results** (out) - **PreTriggerResults**
- **posTransaction** – The **PosTransaction** object

• **void** **PostSuspendTransaction**(ref **PosTransaction** **posTransaction**)

Triggers after the suspended transaction operation executes.

**Parameters**
- **posTransaction** – The **PosTransaction** object

• **void** **PreRecallTransaction**(ref **PreTriggerResults** **results**, ref **PosTransaction** **posTransaction**)

Triggers before the recall-transaction operation executes.

**Parameters**
- **results** (out) - **PreTriggerResults**
- **posTransaction** – The **PosTransaction** object

• **void** **PostRecallTransaction**(ref **PosTransaction** **posTransaction**)

Triggers after the recall-transaction operation executes.

**Parameters**
- **posTransaction** – The **PosTransaction** object
void **OnSuspendTransaction**(RetailTransaction retailTransaction)
Not used.

void **OnRecallTransaction**(RetailTransaction retailTransaction)
Not used.

**ITransactionTriggers interface**
The **ITransactionTriggers** interface contains triggers for transaction operations.

**Triggers**
- void **BeginTransaction**(PosTransaction posTransaction)
  Triggers at the start of a new transaction but after loading the transaction with initialization data, such as the store, terminal number, and date.
  **Parameters**
  - posTransaction – The *PosTransaction* object

- void **PreEndTransaction**(ref PreTriggerResults results, ref PosTransaction posTransaction)
  Triggers at the end of a transaction, before saving the transaction and printing receipts.
  **Parameters**
  - results (out) - PreTriggerResults
  - posTransaction – The *PosTransaction* object

- void **PostEndTransaction**(PosTransaction posTransaction)
  Triggers at the end of a transaction, after saving the transaction and printing receipts.
  **Parameters**
  - posTransaction – The *PosTransaction* object

- void **PreVoidTransaction**(ref PreTriggerResults results, ref PosTransaction posTransaction)
  Triggers before voiding a transaction.
  **Parameters**
  - results (out) - PreTriggerResults
  - posTransaction – The *PosTransaction* object

- void **PostVoidTransaction**(ref PosTransaction posTransaction)
  Triggers after voiding a transaction.
  **Parameters**
  - posTransaction – The *PosTransaction* object
• void **PreReturnTransaction**(ref PreTriggerResults *results*, RetailTransaction *originalTransaction*, ref PosTransaction *posTransaction*)

  Triggers before voiding a transaction.

  **Parameters**
  
  o  *results* (out) - PreTriggerResults
  
  o  *originalTransaction* - The original transaction
  
  o  *posTransaction* – The transaction containing only the selected items to be returned

• void **PostReturnTransaction**(ref PosTransaction *posTransaction*)

  Not used.
## 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.

<table>
<thead>
<tr>
<th>Service interface</th>
<th>Service file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IApplication interface</td>
<td>Application.dll</td>
</tr>
<tr>
<td>IBarcodes interface</td>
<td>Barcodes.dll</td>
</tr>
<tr>
<td>IBankOperations interface</td>
<td>BlankOperations.dll</td>
</tr>
<tr>
<td>ICashChanger interface</td>
<td>CashChanger.dll</td>
</tr>
<tr>
<td>ICCTV interface</td>
<td>CCTV.dll</td>
</tr>
<tr>
<td>ICreditMemo interface</td>
<td>CreateDatabase.dll</td>
</tr>
<tr>
<td>ICurrency interface</td>
<td>Currency.dll</td>
</tr>
<tr>
<td>ICustomer interface</td>
<td>Customer.dll</td>
</tr>
<tr>
<td>IDimension interface</td>
<td>Dimension.dll</td>
</tr>
<tr>
<td>IDiscount interface</td>
<td>Discount.dll</td>
</tr>
<tr>
<td>IDualDisplay interface</td>
<td>DualDisplay.dll</td>
</tr>
<tr>
<td>IEFT interface</td>
<td>EFT.dll</td>
</tr>
<tr>
<td>IEOD interface</td>
<td>EOD.dll</td>
</tr>
<tr>
<td>IGiftCard interface</td>
<td>GiftCard.dll</td>
</tr>
<tr>
<td>IItem interface</td>
<td>Item.dll</td>
</tr>
<tr>
<td>ILoyalty interface</td>
<td>Loyalty.dll</td>
</tr>
<tr>
<td>IPrice interface</td>
<td>Price.dll</td>
</tr>
<tr>
<td>IPrinting interface</td>
<td>Printing.dll</td>
</tr>
<tr>
<td>IPurchaseOrder interface</td>
<td>PurchaseOrderReceiving.dll</td>
</tr>
<tr>
<td>IRFID interface</td>
<td>RFID.dll</td>
</tr>
<tr>
<td>IRounding interface</td>
<td>Rounding.dll</td>
</tr>
<tr>
<td>ISalesInvoice interface</td>
<td>SalesInvoice.dll</td>
</tr>
<tr>
<td>ISalesOrder interface</td>
<td>SalesOrder.dll</td>
</tr>
<tr>
<td>IStockCount interface</td>
<td>StockCount.dll</td>
</tr>
<tr>
<td>IStoreInventoryServices interface</td>
<td>StoreInventoryServices.dll</td>
</tr>
<tr>
<td>ITax interface</td>
<td>Tax.dll</td>
</tr>
<tr>
<td>ITaxProvider interface</td>
<td>ISV to provide DLL name from config file. For details, see ITaxProvider interface.</td>
</tr>
<tr>
<td>ITenderRestriction interface</td>
<td>TenderRestriction.dll</td>
</tr>
<tr>
<td>Trigger interface</td>
<td>Trigger file name</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>IApplicationTriggers interface</td>
<td>ApplicationTriggers.dll</td>
</tr>
<tr>
<td>ICashManagementTriggers interface</td>
<td>CashManagementTriggers.dll</td>
</tr>
<tr>
<td>ICustomerTriggers interface</td>
<td>CustomerTriggers.dll</td>
</tr>
<tr>
<td>IDiscountTriggers interface</td>
<td>DiscountTriggers.dll</td>
</tr>
<tr>
<td>IInfocodeTriggers interface</td>
<td>InfocodeTriggers.dll</td>
</tr>
<tr>
<td>IItemTriggers interface</td>
<td>ItemTriggers.dll</td>
</tr>
<tr>
<td>IPaymentTriggers interface</td>
<td>PaymentTriggers.dll</td>
</tr>
<tr>
<td>ISuspendTriggers interface</td>
<td>SuspendTriggers.dll</td>
</tr>
<tr>
<td>ITransactionTriggers interface</td>
<td>TransactionTriggers.dll</td>
</tr>
</tbody>
</table>