Volume

2

QVS SOFTWARE, INC.

Mobile Point of Sale

Developer's Guide

MOBILE POINT OF SALE

Developer's Guide

© 2002 QVS Software, Inc.
5711 Six Forks Rd. • Suite 300
Phone 919.676.1991 • Fax 919.676.1992

Revision History

Document Number	Date	Author	Comments
	3/30/2005	Donn Machak	Updated Documentation per review.
	3/23/2005	Donn Machak	Updated Documentation for 4.0 Release
	3/19/2001	Jon Akhtar	Initial version

Table of Contents

Chapter 1

"The Problem with Retailers"

The MPOS system has been designed with the need of retailers in mind, however not all retailers are alike.

Most retailers have over time created a highly customized POS solution ideally suited for their business needs. It follows then that these same retailers will want any mobile solution that they implement to be equally customized to meet those same needs. The MPOS application allows for customizations on a per-client basis, while providing basic user-interface and device management features. This frees the developer to write only the code necessary to implement the special features required by an individual client

The Problem

Consider the example of 2 imaginary retailers Retailer A and Retailer B. Retailer A would like the MPOS solution to track the number of credit transactions performed by logging each to a SQL database. Retailer B does not have a SQL server, and does not need to track credit transactions. Retailer B does want to add an extra line to the receipt to indicate that the transaction was performed on the MPOS system. Of course, Retailer A does not want any extra lines added the receipt. With a standard solution the answer would be to add these features into the MPOS application and turn on only the features that each retailer wants. This is a fine idea, but what about retailers C-Z? As the number of customers becomes large, the application would become laden with "special" features that most customers would not use.

The Solution

In order to satisfy both Retailer A and Retailer B, and keep the application manageable, the developer can utilize the MPOS Customization API to augment the functionality of the MPOS application. For Retailer A the developer uses ADOCE to access a remote SQL server on the retailer's ISP, and logs transaction data at the end of each credit transaction. For Retailer B the developer adds a line "*** MPOS ***" to the end of each receipt. Both retailers' needs can be met without modifying the MPOS application at all.

Chapter 2

Introduction to the MPOS Object Model

The MPOS system has been designed to allow for maximum flexibility of implementation. The following is a high level overview designed to prepare you to develop your own customized solutions.

The MPOS object model provides a simple way to change the basic functionality of the MPOS application to meet the specific requirements of the client.

System Components

Figure 1 illustrates the primary components of the MPOS system.

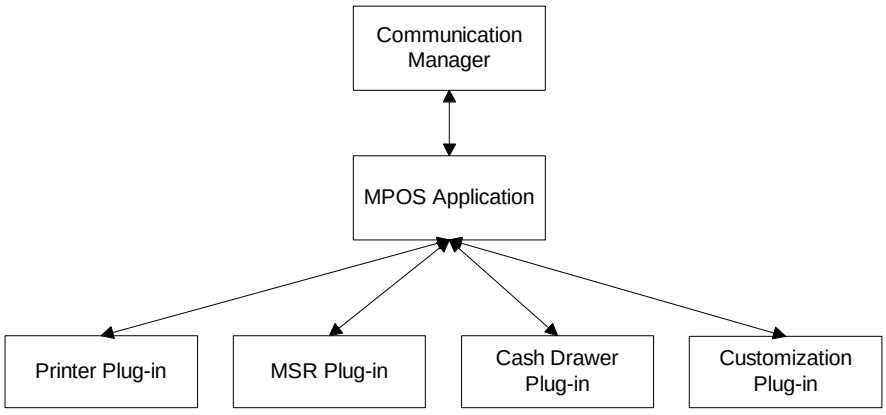

Figure 1 – System Component Block Diagram

The MPOS Application

The primary component, the MPOS application provides the user interface and default behavior for the system. It is contained wholly within the file PPOS_CE.EXE.

The Communications Manager

The Communications Manager handles all communications between the handheld and the 4690 Sales Application running on the Terminal Concentrator. It is contained wholly within the file RPAMFUNCCE.DLL

The Customization Plug-in (Configuration)

The customization plug-in is specific to a particular customer's implementation of the MPOS system. Writing a customized configuration plug-in is the most common task when developing for a new client. Normally the main areas that need coding are the LocalInit() method and the PrinterHook().

The Printer Plug-in

The printer plug-in is specific to a particular device. The MPOS application contains a default plug-in which provides support for running without a printer.

The MSR Plug-in

The MSR plug-in is specific to a particular device. The MPOS application contains a default plug-in which provides support for running without an MSR.

The Cash Drawer Plug-in

The Cash Drawer Plug-in is specific to a particular device. The MPOS application contains a default plug-in which provides support for running without a cash drawer.

Chapter 3

The GenericDriver Object

The GenericDriver interface provides base methods for each device driver object in the MPOS environment.

The GenericDriver interface is inherited by the Configuration, Printer, CashDrawer, and MSR device driver classes. Each of these objects have their own unique methods described in chapters following this common base class. To override the default functionality of the MPOS application the developer will inherit from the GenericDriver class and override one or more of the GenericDriver methods within one or more of the device driver classes (Configuration, Printer, CashDrawer, and MSR).

GetDriverVersionString

Prototype:

```
virtual const TCHAR*    GetDriverVersionString()
```

Remarks:

This method is called to obtain the current version string of the device driver being used.

Notify

Prototype:

```
virtual void Notify(notify msg, long extra)
```

Parameters:

Msg	Notify identifier representing the type of notification. Will be one of the following: notifyWakeup notifyScanLock notifyScanUnlock notifyWindowActivate notifyWindowDeactivate notifyBatteryLow notifyBatteryNorm notifyBatteryCharging notifyAlphaModeOn notifyAlphaModeOff notifyPromptComplete notifyReceptPrinted
Extra	Not used

Remarks:

This method is called when MPOS detects a change that requires the device driver components to be notified. One of the more useful purposes of this method is to be used in the Configuration plugin to know when a transaction receipt has been printed. And as you can see from the list of notifications above you will also know when the alpha entry status has changed and other useful notifications.

Chapter 4

The Configuration Object

For customizing the MPOS application the Customization interface provides the methods necessary to override or augment the application's functionality. It also provides the link to all other objects.

The Configuration allows the developer to override the default functionality of the MPOS application. To do so the developer will inherit from the Configuration class and override one or more of the "hook" methods. There is also a LocalInit() method that is called when the configuration plugin is loaded. Access to the other objects is obtained through a pointer returned by one of the 7 protected access functions, e.g. Application().

LocalInit

Prototype:

virtual bool LocalInit()

Remarks:

This method is called when the application is first loaded. Application and terminal settings can be made at this time.

Sample Code:

```
bool MyCfg::LocalInit()
{
    LOG_SETUP_EX("CConfigFSG::LocalInit");

    // We handle receipt clearing
    Application()->SetAutoReceiptClear(false);

    // Turn off application lockdown for testing
    Application()->SetLockDown(false);

    // Get our device type
    const TCHAR* m_tszDeviceType = Application()->GetDeviceModel();
    TCHAR* pszMatch = _tcsstr(m_tszDeviceType, _T("PPT8800"));
    m_bPPT8800Device = false;
    if (pszMatch != NULL)
    {
        m_bPPT8800Device = true;    // PPT8800 device
    }

    // Get printer plug-in name
    m_tszDeviceType = Application()->GetPluginName(CApplication::pluginPrinter);
    pszMatch = _tcsstr(m_tszDeviceType, NAME_PRN_ZEBRA320);
    m_bPrinterZebra320 = false;
    if (pszMatch != NULL)
    {
        m_bPrinterZebra320 = true;  // Zebra QL320 wireless printer being used
    }

    // We are a full screen POS application
    Rpa()->EnableVDisplay();                // Enable full screen support
    Display()->SetDisplaySize(80, 25);          // 80 columns with 25 rows
    Application()->SetDisplaySource(CApplication::sourceVDisplay);

    // Model 3/4 printer support needed by the POS application
    Rpa()->SetPrinter(CRpa::printerMod34);

    return true;
}
```

StateHook

Prototype:

```
virtual bool StateHook(short sCurState,
                       short sNewState)
```

Parameters:

sCurState	The current state of the 4690 sales application expressed as an short integer 1-255
sNewState	The new state of the 4690 sales application expressed as an short integer 1-255

Remarks:

When the 4690 sales application changes states, this method will be called. The default behavior is to find a corresponding state in the keymap file and display the correct keys for that state. If this behavior is appropriate then the function should return **true** otherwise it should return **false**.

Sample Code:

```
bool CMyCfg::StateHook(short sCurState, short sNewState,
                       short sCurrGroup, short sNewGroup)
{
      // If we are entering the error state, don't change
      // the keys, just put up a message
      if (sNewState == 1)
      {
            Display()->SetText(L"Error");
            return false;
      }

      // for all other states, process the keymap as usual
      return true;
}
```

PrintHook

Prototype:

```
virtual bool PrintHook(const TCHAR* lpszText,
                       short sLF,
                       CRpa::Stations sStation,
                       unsigned long lCommand,
                       unsigned long lFlags)
```

Parameters:

lpszText	The text being printed
sLF	The number of linefeeds following the printed text
sStation	The print station to which the text is being sent. Can be one of: CRpa::stationCR CRpa::stationDI CRpa::stationSJ
lCommand	The command that is sent to the printer (Model 4 printer mode) Can be one of: CRpa::cmdPrintLineStart CRpa::cmdPrintLineEnd CRpa::cmdPrt15CPI CRpa::cmdPrt12CPI CRpa::cmdPrt75CPI CRpa::cmdPrt75CPIDH CRpa::cmdDocEject CRpa::cmdHomeHead CRpa::cmdEmphPrint CRpa::cmdPaperCut CRpa::cmdPrintLineData
lFlags	The flags that are sent to the printer. (Model 4 printer mode) Currently the only possible value is CRpa:: flagLogoPrint

Remarks:

This hook is probably the most used of all the hooks when writing a customized configuration plugin. The main reason it is used the most is the need to possibly print more than one receipt for a credit transaction. To detect that multiple receipts are needed to be printed the user today needs to monitor the text printed and determine from the content that more than one receipt will print. See the simplified code example below that is not complete but shown to help explain the usage of the PrintHook().

Sample Code:

```
bool MyCfg::PrintHook()
{ // Bgn PrintHook
      bool bpReturn = true;       // default is to return and tell MPOS to process
      LOG_SETUP_EX("CConfigFSG::PrintHook");
      if ((m_dwAutoPrint) &&
          (sStation == CRpa::stationCR))
      { // Bgn Autoprint
            //
            // Commands we want to see include the following:
            //
            // cmdPrintLineData
            // cmdPaperCut
            //
            switch (lCommand)
            { // Bgn switch lCommand
            case CRpa::cmdPrintLineData:
                  if (multipleReceiptsNeeded(lpszText)) // Check print line
                        sPaperCutCount += 1;            // extra receipt to print.

                  break;
            case CRpa::cmdPaperCut:
                  --sPaperCutCount;
                  break;
            } // End switch lCommand

            if (sPaperCutCount <= 0)
            {
                  logprintf(LOG_DBG, "Receipt Print\n");
                  bool rc = Receipt()->SendToPrinter(m_cPrinterIP);
            }
      } // End AutoPrint

    return bpReturn;
} // End printHook
```

DisplayHook

Prototype:

```
virtual bool DisplayHook(const TCHAR* lpszText)
```

Parameters:

lpszText	The text being written to the display. This is a 40 character string.

Remarks:

When the 4690 sales application writes to the 2x20 display, this method will be called. The default behavior is write the display data to the operator display. If this behavior is appropriate then the function should return **true** otherwise it should return **false**.

VDisplayHook

Prototype:

```
virtual bool VDisplayHook(long lRow,
                          long lCol,
                          long lLength,
                          const TCHAR* lpszText,
                          COLORREF *iTextArray,
                          COLORREF *iBackArray)
```

Parameters:

lRow	Row number to display text.
lCol	Column number where the text display starts.
lLength	Length of the text to be displayed.
lpszText	The text being written to the display.
iTextArray	Attributes of the text being displayed.
iBackArray	Attributes of the text background being displayed.

Remarks:

When the 4690 sales application writes to the enhanced full screen, this method will be called. The default behavior is write the display data to the full screen display. If this behavior is appropriate then the function should return **true** otherwise it should return **false**.

ErrorHook

Prototype:

virtual bool ErrorHook(CRpa::Errors error)

Parameters:

error	Indicates the error that is being reported by the Communciations Manager. Will be one of: CRpa::errorUnknown CRpa::errorConnectionLost CRpa::errorConnectionFailed CRpa::errorSessionFailed CRpa::errorAttemptingReconnect

Remarks:

StatusHook

Prototype:

virtual bool StatusHook(CRpa::Statuses status)

Parameters:

Status	Indicates the status that is being reported by the Communciations Manager. Will be one of: statusUnknown statusConnectionComplete statusScannerLocked statusScannerUnlocked statusKeyboardLocked statusKeyboardUnlocked statusCashDrawer1Open statusCashDrawer2Open statusConnectionResumed statusMSRLocked statusMSRUnlocked statusManagerKeyOn statusManagerKeyOff statusResyncStart statusResyncComplete statusCtrlrOffline statusCtrlrOnline

Remarks:

When any of the status' listed above occurs this method will be called. The user should almost always return **true** from this routine because the base MPOS code performs numerous tasks when a status changes, e.g. turning the MSR status green when the MSR is enabled.

Sample Code:

```
bool CMyCfg::StatusHook(CRpa::Statuses status)
{
    switch (status)
      {
    case CRpa::statusMSRUnlocked:
            bMSREnabled = true;
            break;
    case CRpa::statusMSRLocked:
            bMSREnabled = false;
            break;
      default:
            break;
      }

    return true; // perform normal handling
}
```

MSRHook

Prototype:

virtual bool MSRHook(const CMsrData* pData)

Parameters:

pData	Pointer to a CMsrData object which contains the information read from the card.

Remarks:

When a card is successfully read, the MSR plug-in will fire an MSREvent that will provide the data from the card in a CMSRData object. The default processing is to send the card data to the 4690 sales application. If this behavior is desired, the function should return **true**, otherwise it should return **false**.

Sample Code:

```
bool CMyCfg::MSRHook(const CMsrData* pData)
{
      // If the first byte of the track 1 data is "7" then
      // it is a manager key card, so turn on the manager key, but don't
      // send the card data to the sales application
      if (pData->GetTrack(1)[0] == "7")
      {
            Rpa()->SetManagerKey(true);
            return false;
      }
      else
            return true;
}
```

ScanHook

Prototype:

```
virtual bool ScanHook(CRpa::Labeltypes type, const TCHAR* lpszData)
```

Parameters:

Type	Indicates the type of barcode label that was scanned. Will be one of the following: labeltypeUnknown labeltypeUPC_E1 labeltypeUPC_E0 labeltypeUPC_A labeltypeEAN8 labeltypeEAN13 labeltypeCodabar labeltypeMSI labeltypeCode39 labeltypeD2of5 labeltypeI2of5 labeltypeCode11 labeltypeCode93 labeltypeCode128 labeltypeCode32 labeltypeIATA2of5 labeltypeEAN128
lpszData	Pointer to the barcode data scanned.

Remarks:

When a user scans a barcode this routine will get called. The default processing is to send the scan data to the 4690 sales application. If this behavior is desired, the function should return **true**, otherwise it should return **false**.

This are is where a user can have unique barcodes scanned to perform specific non-POS tasks. An example would be that a developer may want to enable or disable certain features in their test environment and one way they can accomplish that is by scanning a unique barcode that is interpreted via this hook. Be careful that you always return **true** for normal POS transaction processing.

FcodeHook

Prototype:

virtual bool FcodeHook(short sFcode)

Parameters:

sFcode	The keyboard function code that is going to be sent to the 4690 Sales Application.

Remarks:

This method is called just before the keyboard function code is sent to the 4690 Sales Application. The default behavior is to send any function code less than 255. If this behavior is desired then this method should return **true**, otherwise it should return **false** to prevent the function code from being sent.

Sample Code:

```
bool CMyCfg::FcodeHook(short sFcode)
{
      bool bReturn = true;    // default is to let MPOS process the key
      // Filter out function code 200, and just display
      // a message – don't pass it on to the sales appl.
      if (sFcode == 200)
      {
            Display()->SetText(L"Found");
            bReturn = false; // Don't send to sales application
      }

      // Turn alpha entry off so CLEAR key gets through
      // In this example the CLEAR key is defined as 73 (x49)
      if (sCurrState == STATE_CLEAR &&
          sFcode == KEY_CLEAR &&
          Rpa()->GetStateAlpha())
      {
            Rpa()->SetStateAlpha(0);
      }
      return bReturn;
}
```

FieldLevelInputEvent2Hook

Prototype:

```
virtual bool FieldLevelInputEvent2Hook(long lMaxLen,
                                       long lEchoInput,
                                       long lFirstDisplayCol,
                                       long lLastDisplayCol,
                                       long lStateNum,
                                       BOOL bAlphaEntry,
                                       long lTransitionFlag)
```

Parameters:

lMaxlen	Maximum length of input.
lEchoInput	Is input echoed or not. Zero says no.
lFirstDisplayCol	The first column where input is displayed.
lLastDisplayCol	Last column where input is displayed.
lStateNum	Current POS application state number.
bAlphaEntry	Alpha entry enabled or disabled.
lTransitionFlag	Was this event caused by a state transition or not.

Remarks:

This method is only useful for a POS application that uses the enhanced full screen feature. It is useful in the full screen applications in allowing the programmer to know where the focus of data input is in case the programmer needs to highlight fields.

Chapter 5

The Application Object

The application object provides methods to allow the developer to modify MPOS application functionality.

SetTitle

Prototype:

```
virtual void SetTitle(const TCHAR* lpszTitle)
```

Parameters:

lpszTitle	String containing the new application title

Remarks:

Call this function to set the text on the title bar of the MPOS application.

GetCurrState

Prototype:

```
virtual short GetCurrState()
```

Remarks:

Returns the current input state of the 4690 sales application.

GetPrevState

Prototype:

```
virtual short GetPrevState()
```

Remarks:

Returns the previous input state of the 4690 sales application.

SetTerminalID

Prototype:

```
virtual void SetTerminalID(short sTermID)
```

Parameters:

STermID	The new terminal ID

Remarks:

Sets the terminal ID to be used the next time the MPOS application connects to the terminal concentrator.

This method should be used in the Init method of the configuration object but is normally assigned via the MPOS application.

The default value is the last octet of the IP address modulo 100.

GetTerminalID

Prototype:

```
virtual short GetTerminalID()
```

Remarks:

Gets the terminal ID of the active session. This may differ from the terminal ID returned from the application object.

Terminate

Prototype:

```
virtual void Terminate()
```

Remarks:

Terminate the MPOS application.

SetLockDown

Prototype:

```
virtual void SetLockDown(bool bLockDown)
```

Parameters:

bLockDown	The new value for this property.

Remarks:

The application can be "locked down" to prevent users from running other programs. When the application is locked down, there will be no start menu or Ok button.

The default value is true.

AllowDeviceSleep

Prototype:

```
virtual void AllowDeviceSleep(bool bAllowSleep)
```

Parameters:

bAllowSleep	The new value for the property

Remarks:

Set this to allow/disallow the device to enter sleep mode.

The default value is true

PowerOff

Prototype:

```
void PowerOff()
```

Remarks:

Power the device off.

SetAutoReceiptClear

Prototype:

```
virtual void SetAutoReceiptClear(bool bAutoClear)
```

Parameters:

BAutoClear	The new value for the AutoReceiptClear property

Remarks:

The MPOS application will automatically clear the onscreen receipt when it successfully completes a print request. This feature may be enabled or disabled by using this method.

The default value is true.

GetScannerHandle

Prototype:

```
virtual unsigned long GetScannerHandle()
```

Remarks:

Returns the handle for the scanner.

GetDeviceModel

Prototype:

```
const TCHAR* GetDeviceModel()
```

Remarks:

Returns the assigned name for the current handheld device. An example would be that the 8146 device will return a name that contains the string PPT8800 in it. This is useful if you are running your customized configuration plugin on more than one device and need to perform certain code based on the type of device you are running. This does away with the need to have "#ifdef" code in your source. You could set a flag to identify the device type and execute code based off of the flag.

GetPluginName

Prototype:

```
const TCHAR* GetPluginName(plugins p)
```

Parameters:

p	The plugin whose name is being requested. Must be one of: pluginConfiguration pluginPrinter pluginMSR pluginCashDrawer pluginSigCap

Remarks:

Returns the name of the specified plugin. This is useful if different behavior is required for different hardware configurations.

GetLayout

Prototype:

```
const TCHAR* GetLayout()
```

Remarks:

Returns the layout name that is currently in effect. The layout name is defined in the XML configuration file (also called the keymap file).

GetPrevLayout

Prototype:

```
const TCHAR* GetPrevLayout()
```

Remarks:

Returns the previous layout name that was in effect before the current layout. The layout name is defined in the XML configuration file (also called the keymap file).

GetView

Prototype:

```
const TCHAR* GetView()
```

Remarks:

Returns the current view name that is in effect. The view name is defined in the XML configuration file (also called the keymap file).

RefreshAll

Prototype:

```
virtual void RefreshAll()
```

Remarks:

Refreshes the current layout and view.

SetLayout

Prototype:

```
virtual void SetLayout(const TCHAR*strLayoutID, const TCHAR* strViewID)
```

Parameters:

strLayoutID	Layout name.
strViewID	View name.

Remarks:

Sets the current view and layout as defined in the passed parameters. The layout name and view name is defined in the XML configuration file (also called the keymap file).

SetView

Prototype:

```
virtual void SetView(const TCHAR* strViewID)
```

Parameters:

strViewID	View name.

Remarks:

Sets the current view as defined in the passed parameters. The layout name and view name is defined in the XML configuration file (also called the keymap file).

GetAppPath

Prototype:

```
const TCHAR* GetAppPath()
```

Remarks:

Returns the path of where the MPOS application is currently running from.

DoScanCommand

Prototype:

```
virtual void DoScanCommand(const TCHAR* lpszCommand)
```

Parameters:

lpszCommand	Scan command for MPOS to process.

Remarks:

Tells MPOS to process the scan command as if the data were scanned in from a barcode. An example would be to send the string "MPOSExit" to tell MPOS to exit.

SendKey

Prototype:

```
virtual long  SendKey(short sFcode)
```

Parameters:

sFcode	The function code to send to the 4690 sales application

Remarks:

Use this method to send a function code to the POS application. Value can be from 1 to 255 and should be a valid function code that the POS application knows about.

ScanDisable

Prototype:

```
virtual void ScanDisable()
```

Remarks:

Use this method to disable the scanner.

ScanEnable

Prototype:

```
virtual void ScanEnable()
```

Remarks:

Use this method to enable the scanner.

GetDisplay

Prototype:

```
virtual CDisplay* GetDisplay(const TCHAR* lpszDisplayName)
```

Parameters:

lpszDisplayName	NULL or the name of the CDisplay object wanted.

Remarks:

Returns a pointer to the CDisplay object requested.

SetDisplaySource

Prototype:

```
virtual void SetDisplaySource(source s)
```

Parameters:

s	The display source. Can be: sourceOperatorDisplay (2x20) sourceVDisplay (full screen)

Remarks:

Use this method to set the display source properly. This will be executed in the LocalInit() call of the customized configuration plugin. See the LocalInit() sample code for usage.

The default value is set to sourceOperatorDisplay (2x20 POS application).

SetRFOutOfRange

Prototype:

```
virtual void SetRFOutOfRange(long lThreshold)
```

Parameters:

lThreshold	Set the value to be used by MPOS for when it is determined that the hand held device is out of range to the access point. Valid value is 1 to 100.

Remarks:

Use this method to change the default value for when MPOS will determine that the hand held device is out of range to the access point.

The default value is set to 20.

MessageBox

Prototype:

```
int  MessageBox(const TCHAR* lpszPrompt, UINT nType)
```

Parameters:

lpszPrompt	Text to be prompted in the message box.
nType	Type of message box to be prompted.

Remarks:

Returns the value determined from the message box routine.

Chapter 6

The RPA Object

The RPA object provides methods to control the terminal client aspect of the application. It is the primary interface with the 4690 sales application.

SendKey

Prototype:

```
virtual long  SendKey(short sFcode)
```

Parameters:

sFcode	The function code to send to the 4690 sales application

Remarks:

Use this method to send a function code to the POS application. Value can be from 1 to 255 and should be a valid function code that the POS application knows about.

SendScan

Prototype:

```
virtual long  SendScan(Labeltypes ScanType, const TCHAR* lpszData)
```

Parameters:

ScanType	The barcode type. Must be one of:
lpszData	The scan data

Remarks:

This method is normally used via the ScanHook() and is needed to sometimes override scan data sent to the POS application by MPOS. There have been instances where the scanner will send in a certain barcode type that the POS application does not support. The ScanHook() allows the programmer to interpret the scan that just happened and possibly change the barcode type and or data so that it is accepted properly by the POS application.

SendMSR

Prototype:

```
virtual long   SendMSR(const char* lpszTk1, const char* lpszTk2, const char* lpszTk3)
```

Parameters:

lpszTk1	The track 1 data
lpszTk2	The track 2 data
lpszTk3	The track 3 data

Remarks:

Use this method to override MSR data that is sent by MPOS. MPOS will send in all the proper MSR data to the POS application but in some instances a user may want to override the MSR data and only send in certain track data to the POS application.

GetManagerKey

Prototype:

```
virtual bool  GetManagerKey()
```

Remarks:

Gets the status of the manager key. True returned if the manager key status is on else returns false if the manager key is turned off.

SetManagerKey

Prototype:

```
virtual void  SetManagerKey(bool bNewValue)
```

Parameters:

bNewValue	The new value for this property

Remarks:

Use this method to turn the manager key on or off. If the POS application requires the user to turn the manager key on and off you would use this method along with possibly defining a key in your keymap file (XML configuration file) that allows a user to simulate turning the manager key on or off. Or if your customer allows you can always simply send in that the manager key is on so that no manager key required errors occur.

GetStateAlpha

Prototype:

```
short GetStateAlpha()
```

Remarks:

This method tells you if the current state is in alpha entry mode or not. A zero value says no alpha entry is enabled while a return code value of 1 says that alpha entry is enabled for the current state.

SetStateAlpha

Prototype:

```
virtual void  SetStateAlpha(short bNewAlphaValue)
```

Parameters:

bNewAlphaValue	The new value for this property

Remarks:

Use this method to tell the POS application to enter or exit alpha entry mode.

A common area that this method is used for is when a user hits the CLEAR key and instead of the CLEAR key being sent to the POS application the user sees an "I" displayed. Most applications have the function code 73 defined as the CLEAR key which is also the alpha letter "I" if alpha entry is supported by the POS application. In this instance the user should probably intercept the function code being sent via the FcodeHook() and turn alpha entry off if you want the CLEAR key to go through to the POS application rather than the letter "I."

See the sample code in the FcodeHook() for reference.

SetPrinter

Prototype:

```
virtual void  SetPrinter(Printertypes printer)
```

Parameters:

bNewValue	Printer type. Values can be: printerMod2 printerMod34

Remarks:

Used to set what type of printer that the POS application is using.

SetDocumentInsert

Prototype:

```
virtual bool  SetDocumentInsert( bool bNewValue )
```

Parameters:

bNewValue	The new value for this property

Remarks:

Notifies the 4690 sales application of a change in that status of the document insert station.

GetScannerLock

Prototype:

```
virtual bool  GetScannerLock()
```

Remarks:

Gets the lock state of the scanner.

GetKeyboardLock

Prototype:

```
virtual bool  GetKeyboardLock()
```

Remarks:

Gets the lock state of the keyboard.

GetMSRLock

Prototype:

```
virtual bool  GetMSRLock()
```

Remarks:

Gets the lock state of the MSR.

GetTerminalID

Prototype:

```
short GetTerminalID()
```

Remarks:

Gets the terminal ID of the active session. This may differ from the terminal ID returned from the application object.

SetFldLvlInput

Prototype:

```
virtual void SetFldLvlInput(bool bFldLvl)
```

Remarks:

Use this method to enable or disable field level input.

EnableVDisplay

Prototype:

```
virtual void  EnableVDisplay()
```

Remarks:

Use this method to enable VDisplay usage which means the POS application is using the enhanced full screen display. See the LocalInit() sample code for usage.

Chapter 7

The Receipt Object

The receipt object provides the interface to the on-screen receipt.

Clear

Prototype:

```
virtual void Clear()
```

Remarks:

Clear the contents of the on-screen receipt and associated print buffers.

PrintLine

Prototype:

```
virtual void PrintLine(const TCHAR* lpszLine, CRpa::Stations station)
```

Parameters:

lpszLine	The text to send to the receipt
station	Station to output the text to print. Values can be: stationCR stationSJ stationDI

Remarks:

Print a line to a receipt station.

PrintLine

Prototype:

```
virtual void PrintLine(const TCHAR* lpszLine,
               const TCHAR* lpszLineUI,
               CRpa::Stations station)
```

Parameters:

lpszLine	The text to send to the receipt
lpszLineUI	The text to send to the receipt object online recipt view.
station	Station to output the text to print. Values can be: stationCR stationSJ stationDI

Remarks:

Print a line to a receipt station.

GetLine

Prototype:

```
virtual void GetLine(int nLine, TCHAR* lpszLine)
```

Parameters:

nLine	The line number to get
lpszLine	The buffer in which to store the string

Remarks:

Get a specific line on the receipt.

InsertCut

Prototype:

```
virtual void InsertCut()
```

Remarks:

Place a paper cut command after the last line.

InsertLogo

Prototype:

```
virtual void InsertLogo()
```

Remarks:

Place a logo command after the last line.

InsertDI

Prototype:

virtual void InsertDI(const TCHAR* lpszPrompt)

Parameters:

lpszPrompt	The prompt to display to the user.

Remarks:

Use this method to store the document insert prompt. When the print actually occurs, the application will display this prompt and wait for the user to press the CLEAR key before printing to the document insert station.

Sample Code:

```
bool CMyCfg::DisplayHook(const TCHAR* lpszText,
                         short sCurState,
                         short sCurrGroup)
{
      if (wcsncmp(lpszText, L"INSERT APPLICATION", 17) == 0)
      {
            Rpa()->SetDocumentInsert(true);
            Rpa()->SendKey(fcodeClear);

            if (Printer()->HasDI())
                  Receipt()->InsertDI(lpszText);

            return false;
      }

      return true;
}
```

InsertBarcode

Prototype:

```
virtual void InsertBarcode(CRpa::Labeltypes type, TCHAR* lpszData)
```

Parameters:

Type	Barcode type label.
lpszData	Barcode data.

Remarks:

Use this method to store a barcode into the print receipt.

InsertCashdrawer

Prototype:

```
virtual void InsertCashdrawer(short sNum)
```

Parameters:

sNum	Cash drawer number.

Remarks:

Use this method to store …

SendToPrinter

Prototype:

```
virtual void SendToPrinter(const char* lpszHost)
```

Parameters:

lpszHost	A string that contains the host IP address, or other information as required by the printer plugin.

Remarks:

Print the sales receipt.

Chapter 8

The Display Object

The display object provides the interface to the on-screen display of the MPOS application.

SetText

Prototype:

```
virtual void SetText(TCHAR* lpszText)
```

Parameters:

lpszText	The text to place on the display.

Remarks:

Based on the type of display being used this text length may vary.

GetText

Prototype:

```
virtual void GetText(TCHAR* lpszBuffer, int nLen)
```

Parameters:

lpszBuffer	The buffer in which to place the contents of the display. Should be a buffer large enough to store 40 characters
nLen	The size of the lpszBuffer

Remarks:

Returns the first nLen bytes of the display text.

GetTextRect

Prototype:

```
virtual void GetTextRect(TCHAR* lpszBuffer,
                  int iLeft,
                  int iTop,
                  int iRight,
                  int iBottom)
```

Parameters:

lpszBuffer	The buffer in which to place the contents of the display. Should be a buffer large enough to store all characters requested.
iLeft	Top left column
iTop	Top row
iRight	Bottom right column
iBottom	Bottom row

Remarks:

Returns the text based on the rectangle dimensions.

GetTextAttrRect

Prototype:

```
virtual void GetTextAttrRect(COLORREF* pBufferFore,
                      COLORREF* pBufferBack,
                      int iLeft,
                      int iTop,
                      int iRight,
                      int iBottom)
```

Parameters:

pBufferFore	Foreground attributes of text requested.
pBufferBack	Background attributes of text requested.
iLeft	Top left column
iTop	Top row
iRight	Bottom right column
iBottom	Bottom row

Remarks:

Returns the text attributes based on the rectangle dimensions.

SetDisplaySize

Prototype:

```
virtual void SetDisplaySize(int x, int y)
```

Parameters:

X	Number of columns (max 80)
Y	Number of rows (max 25)

Remarks:

Used when the enhanced full screen display is written to by the POS application. This call will be made in the LocalInit() method of the customized configuration plugin.

GetDisplaySize

Prototype:

```
SIZE GetDisplaySize()
```

Parameters:

SIZE	Returns the size of the display that we are using.

Remarks:

Used to determine our display size.

SetViewport

Prototype:

```
virtual void SetViewport(int iLeft, int iTop, int iRight, int iBottom)
```

Parameters:

iLeft	Top left column
iTop	Top row
iRight	Bottom right column
iBottom	Bottom row

Remarks:

Used when the enhanced full screen display is used and we want to show only a portion of the full display if our device is of the ¼ VGA display size.

GetViewportSize

Prototype:

```
SIZE GetViewportSize()
```

Parameters:

SIZE	Returns the size of the display that we are using.

Remarks:

Used to determine the display size of the current viewport.

PutText

Prototype:

```
virtual void PutText(int row, int col, const TCHAR* lpszText)
```

Parameters:

Row	Place the text on this row.
Col	Place the text starting at this column.
Lpsztext	Text to be displayed.

Remarks:

Used when the enhanced full screen display is written to by the POS application. Use this method to output text to a certain row and column.

PutTextWithAttributes

Prototype:

```
virtual void PutTextWithAttributes(int row,
                                   int col,
                                   const TCHAR* lpszText,
                                   COLORREF* iTextArray,
                                   COLORREF* iBackArray)
```

Parameters:

Row	Place the text on this row.
Col	Place the text starting at this column.
Lpsztext	Text to be displayed.
iTextArray	Attributes to apply to the text.
iBackArray	Attributes to apply to the background of the text.

Remarks:

Used when the enhanced full screen display is written to by the POS application. Use this method to output text to a certain row and column with attributes used.

ClearBuffer

Prototype:

```
virtual void ClearBuffer()
```

Remarks:

Used to clear the display buffer to blanks.

ClearViewport

Prototype:

```
virtual void ClearViewport()
```

Remarks:

Used to clear the current viewport in use to blanks.

SetLogFont

Prototype:

```
virtual void SetLogFont(const LOGFONT* pLogFont)
```

Parameters:

pLogFont	Pointer to the log font to be used by the display.

Remarks:

Used to set the log font to be used for the display.

Chapter 9

The Printer Object

The printer object provides the interface to the physical print device

BeginReceipt

Prototype:

```
virtual long BeginReceipt(const char* lpszHost)
```

Parameters:

lpszHost	This would be the IP host address (IP address, e.g. 111.222.333.444) of the wireless printer being used.

Remarks:

This is the first method called when the SendToPrinter() method is called via the Receipt object.

PrintLine

Prototype:

```
virtual long PrintLine(char* lpszData, int nLen, Stations station)
```

Parameters:

lpszData	Text receipt data to be printed.
station	Station to print to which is normally the customer receipt station.

Remarks:

This method is called for each print line executed by the POS application.

PrintBarcode

Prototype:

```
virtual long PrintBarcode(CRpa::Labeltypes type, const char* lpszData)
```

Parameters:

Type	Barcode type to print. Values can be one of the following: labeltypeUPC_E1 labeltypeUPC_E0 labeltypeUPC_A labeltypeEAN8 labeltypeEAN13 labeltypeCodabar labeltypeMSI labeltypeCod39 labeltypeD2of5 labeltypeI2of5 labeltypeCode11 labeltypeCode93 labeltypeCode128 labeltypeCode32 labeltypeIATA2of5 labeltypeEAN128
lpszData	Barcode data to print.

Remarks:

This method is called to print barcodes.

PrintLogo

Prototype:

```
virtual long PrintLogo(bool bInit)
```

Parameters:

bInit	This flag is used if the PrintLogo() is the first call made and we want to issue the BeginReceipt() call before the logo is inserted.

Remarks:

This method is called to print a logo.

EndReceipt

Prototype:

```
virtual long EndReceipt()
```

Remarks:

This is the last method called when the SendToPrinter() method is called via the Receipt object.

HasDI

Prototype:

```
virtual bool HasDI()
```

Remarks:

This method tells MPOS if the attached MPOS printer has support for the document insert (DI) station or not. If so, then all document insert data will be sent to the attached MPOS printer if needed.

Returns true if the printer supports DI prints else the printer driver returns false.

HasSJ

Prototype:

```
virtual bool HasSJ()
```

Remarks:

This method tells MPOS if the attached MPOS printer has support for the journal (SJ) station or not. If so, then all journal data will be sent to the attached MPOS printer if needed.

Returns true if the printer supports SJ prints else the printer driver returns false.

HasCut

Prototype:

```
virtual bool HasCut()
```

Remarks:

This method tells MPOS if the attached MPOS printer has support for the CUT command or not. If so, then all CUT commands will be sent to the attached MPOS printer if needed.

Returns true if the printer supports the CUT command else the printer driver returns false.

SetDIOpen

Prototype:

```
virtual long SetDIOpen(bool bIsOpen)
```

Parameters:

bIsOpen	True if MPOS wants to open the DI station to start printing to the DI. False if the DI station is open and MPOS wants to print to a different station.

Remarks:

This method tells the associated printer driver if MPOS wants to open or close the DI station for printing. This method is only used if the printer driver has DI support (see HasDI() method).

CutReceipt

Prototype:

```
virtual long CutReceipt()
```

Remarks:

This method will cut the receipt if the attached printer says it supports cutting the receipt (see HasCut() method). If no receipt cutting support is available then this call will not be made by MPOS.

PrintInverted

Prototype:

```
virtual long PrintInverted(bool bInvert)
```

Parameters:

bInvert	True if MPOS wants the attached printer to print the receipt inverted. False is the default value.

Remarks:

This method tells the associated printer driver if MPOS wants the printer driver to print the text inverted. The default value is to not print inverted.

WantsESC

Prototype:

```
virtual bool WantsESC()
```

Remarks:

This method is called to ask the attached printer driver if it supports and wants all the printer escape sequences sent to it. These escape sequences would be the special printer commands sent to bold the text, print the text in double height, etc.

Chapter 10

The MSR Object

The MSR object provides the interface to the physical MSR device.

Start

Prototype:

```
virtual bool Start()
```

Remarks:

Start the MSR device so that it can be used.

Returns true if the operation was successful or false if the MSR is already started.

End

Prototype:

```
virtual bool End()
```

Remarks:

Stop the MSR device from being used.

Returns true if the operation was successful.

Pause

Prototype:

```
virtual bool Pause()
```

Remarks:

Pause reading of the MSR

Returns true if the operation was successful.

Resume

Prototype:

```
virtual bool Resume()
```

Remarks:

Resume/start reading of the MSR

Chapter 11

The CashDrawer Object

Cash drawer management is supplied by the MPOS application. If developers need to manage the cash drawer directly, the cash drawer object provides the necessary interface with the device.

Open

Prototype:

```
virtual void Open(int nDrawerNumber, const char* lpszHost = NULL)
```

Parameters:

nDrawerNumber	The drawer that should be opened, usually this is either 1 or 2.
lpszHost	A string that contains the host IP address, or other information as required by the cashdrawer plugin.

Remarks:

If no plugin is installed this method will do nothing.

IsOpen

Prototype:

```
virtual bool IsOpen(int nDrawerNumber, const char* lpszHost = NULL)
```

Parameters:

nDrawerNumber	The drawer that should be checked for an open status, usually this is either 1 or 2.
lpszHost	A string that contains the host IP address, or other information as required by the cashdrawer plugin.

Remarks:

This method returns true if the requested drawer is open else a false return is given.

If no plugin is installed this method will alternate between returning true and returning false.