FuelDirect

fuel control solution featuring PI FuelNet connectivity

Dispenser Development Guide

- General Theory of Development
- Fuel Direct — OLE2 Server
- Generic Pump Simulator Program

Progressive International Electronics

2422 Atlantic Avenue, Raleigh, North Carolina 27604

Progressive International Electronics' patented technologies ensure state of the art control capabilities for our worldwide market. Our staff is highly trained and ready to assist you with everything from product development to providing solutions to POS dispenser control.

MEMBER PEI PETROLEUM EQUIPMENT INSTITUTE

phone 919 . 821.1323
fax 919 . 821.1325
e-mail corp@pie-corp.com
web site www.pie-corp.com

DISPENSER DEVELOPMENT GUIDE

PURPOSE

It is the intention of Progressive International Electronics to provide all the information necessary about its own products and software to aid companies who are interfacing with Progressive products. The Dispenser Development Guide is a comprehensive document which offers these explanations, as well as data pertaining to interfacing with other petroleum equipment. For more detail on any product not manufactured by PIE, always refer to that product's accompanying documentation.

EXPLANATION OF DOCUMENT FORMAT

The following documentation standards are applied throughout the Progressive International's Dispenser Development Guide.

- Comments are noted in *italics*.
- Variable data formats are represented by X(s).
- Command format text is case sensitive, and is represented as such throughout this document.

NOTICE

This document may contain typographical errors or technical inaccuracies. Progressive International Electronics reserves the right to revise and improve its equipment, as well as this document, as required. This publication details the process of interfacing to our products at this time, and may not accurately describe this process at all times in the future. Specifications are subject to change without notice. Software revisions are documented in the CD which accompanies this manual prior to making revisions to the hard copy.

COPYRIGHT

Dispenser Development Guide

FuelDirect

HISTORY OF DOCUMENTATION CHANGES & REVISIONS

Version 1.0 — October 10, 2004

Original Issue

Version 2.0

Reader Methods, Keypad Configuration – added Tokheim Premier C Series In-Site keypad layout

Version 3.0 – April 2005

Dispenser Methods – added Dispenser Information
Reader Methods – added Reader Information
Tank Monitor Methods – added Tank Monitor Start Report, Tank Monitor Report Status and Tank Monitor Report

Version 4.0 – August 2005

Reader Methods – added barcode sequence to PrintControl
Reader Methods – further defined PacketQueInput Method
Reader Methods – further defined PacketQueOutput Method

Version 4.1 – April 2006
Minor corrections

Version 4.2 – July 2007
Added Car Wash Methods

Dispenser Development Guide

FuelDirect

CONTENTS

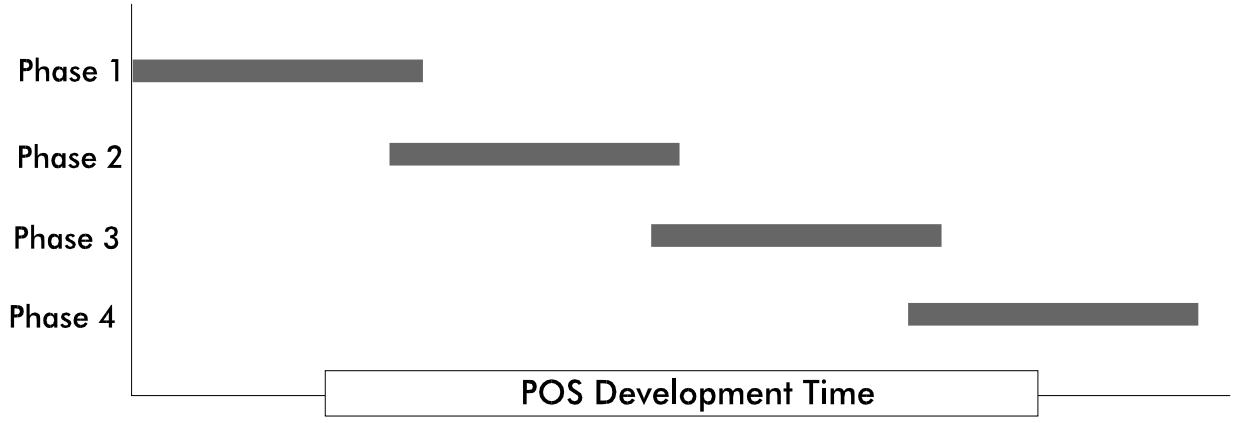
GENERAL THEORY OF DEVELOPMENT
OVERVIEW OF APPLICATION DEVELOPMENT

Progressive International Electronics'FuelDirect controller provides all the functions necessary for a Point of Sale (POS) system to seamlessly control fueling dispensers and card readers.

When designing a POS system to run a fueling site, the process can be broken down into phases. A diagram of the four phases of POS development is shown below. Although this is only a general guideline of the design process, the order shown is very important. As an example, it is essential that dispenser control is successfully implemented and fully tested before any card reader control is attempted. With this sequence in mind, it is important to realize that it is not necessary to complete all phases before selling your system. This is just a recommended order of development that coincides with a staged marketing plan.

From observing many POS developments over the years, we have seen that each phase of this process typically takes six months to a year. Obviously, each stage can be completed more quickly when more resources are devoted to the project.

The chart below illustrates how the four phases of the project should progress.



FOUR PHASES OF POS DEVELOPMENT

Phase 1: Data Collection

Gain an understanding of how fueling sites operate. Visiting several different types of fueling sites and monitoring their operation will be very helpful. Pay particular attention to these two areas:

- In-store operations — Requirements are a combination of management and operator functionality. Managers want systems that run efficiently and in concert with their entire business information technology. They also require systems that provide security and accountability of their store merchandise and fuel. Designing a flexible database/reporting system will help satisfy the many different requirements managers place on a POS. On the other hand, POS operators need a friendly interface that is not intimidating. Simple screens, well laid-out keyboards, fast, user-friendly printers and quick training cycles are a must from an operator standpoint.

- Customer at the dispenser — Can and create operation scenarios that will boggle the mind! Observing what takes place at the gas island will prepare you for the worst — and enable you to create an application that takes the worst-case scenario into consideration. Customers can be impatient and their actions, such as pushing the same key repeatedly or jiggling the dispenser handle, demonstrate just a few of the problems that can be created.

How your POS deals with these requirements and potential challenges will determine its market value.

Phase 2: Application Development — POS and Dispensers

- Get POS application communicating successfully with FuelDirect — OLE2 interface, which is a Windows-based driver. *Refer to PIE's Application Development Tools later in this section for more information.*

- Get dispensers running successfully in lab using PIE's and software dispenser simulator. At no additional charge, PIE provides software to develop, test and demonstrate the FuelDirect — OLE2 controllers. *See PIE's Application Development Tools.*

- Get dispensers running successfully in lab on test dispensers (purchased from dispenser manufacturers such as Gilbarco, Wayne, etc.). This will allow the closest simulation of a fueling site without being on location. To prevent a great deal of on-site debugging, we highly recommend that this step be implemented.

- Secure beta test site to test dispenser application. Many petroleum marketers are willing to supply a beta site, since both the developer and the petroleum marketer will benefit.

- Market dispenser control version of POS system.

Phase 3: Application Development — Card Reader/Credit Card

- Get readers running successfully in lab using PIE's and software dispenser simulator. At no additional charge, PIE provides software to develop, test and demonstrate the FuelDirect — OLE2 controllers. *See PIE's Application Development Tools.*

- Get readers running successfully in lab on test dispensers. (Test dispensers may be purchased from dispenser manufacturers such as Gilbarco or Wayne.)

- Bank cards such as VISA and MasterCard are the next logical POS enhancement, since many card verification networks provide approval for bank cards. Carefully consider which of the many card verification networks best suits your needs. As an example, some of the major oil credit cards are supported by these networks, and others are not. Also, some major oil credit cards have direct verification networks which must be used for approval and require a POS certification which is very selective. A number of design issues must be considered when implementing a card system. The card reader service and credit card network access/approval method should be carefully developed.

- Secure beta test site to test dispenser/credit card application.

- Market dispenser/reader version of POS system.

Phase 4 (optional): Application Development — Card Reader/Debit Card

- Implement debit card operation. This is undoubtedly the most difficult aspect of the total POS solution to implement due to encryption schemes employed for card security. Three primary system design issues need to be addressed — card reader service, debit card network access, and data encryption.

- Configure card readers. Card readers can be configured to support DES, the industry standard encryption method. Note that, due to the additional expense of adding this option, some dispenser card readers do not have DES encryption keypads. Contact the dispenser manufacturer to acquire the security module, security module protocol and dispenser/reader simulator with DES keyboard for testing.

  FuelDirect — OLE2 utilizes a pass-thru scheme to transfer all security module messages. In so doing, our FuelDirect makes no attempt at decoding the data, ensuring data security at the controller level. Note that certification on a DES network requires significant expenditures of time and money, since debit networks must ensure successful performance in all areas.

- Secure beta test site to test dispenser and credit/debit application.

- Market dispenser/reader version of POS with debit capabilities.

PIE'S APPLICATION DEVELOPMENT TOOLS

Progressive International supplies FuelDirect — OLE2 documentation which defines the interface protocol and facilitates the process of attaching directly to our FuelDirect — OLE2 server. POS systems using Windows operating systems are ideally suited for this style of interface.

To aid in the development of the POS to FuelDirect — OLE2 server, PIE provides a dispenser simulator at no charge.

GENERAL THEORY OF DEVELOPMENT
FUNDAMENTAL DEVELOPMENT PROCEDURES

These fundamental procedures are necessary to properly operate PIE's FuelDirect controller:

- The FuelDirect must be regularly polled for fuel position information and queue count (at least once a second). Use the Status Flow Chart provided with PIE's documentation to interpret status and act accordingly. *See Example Flow Chart in this section for start-up process.*

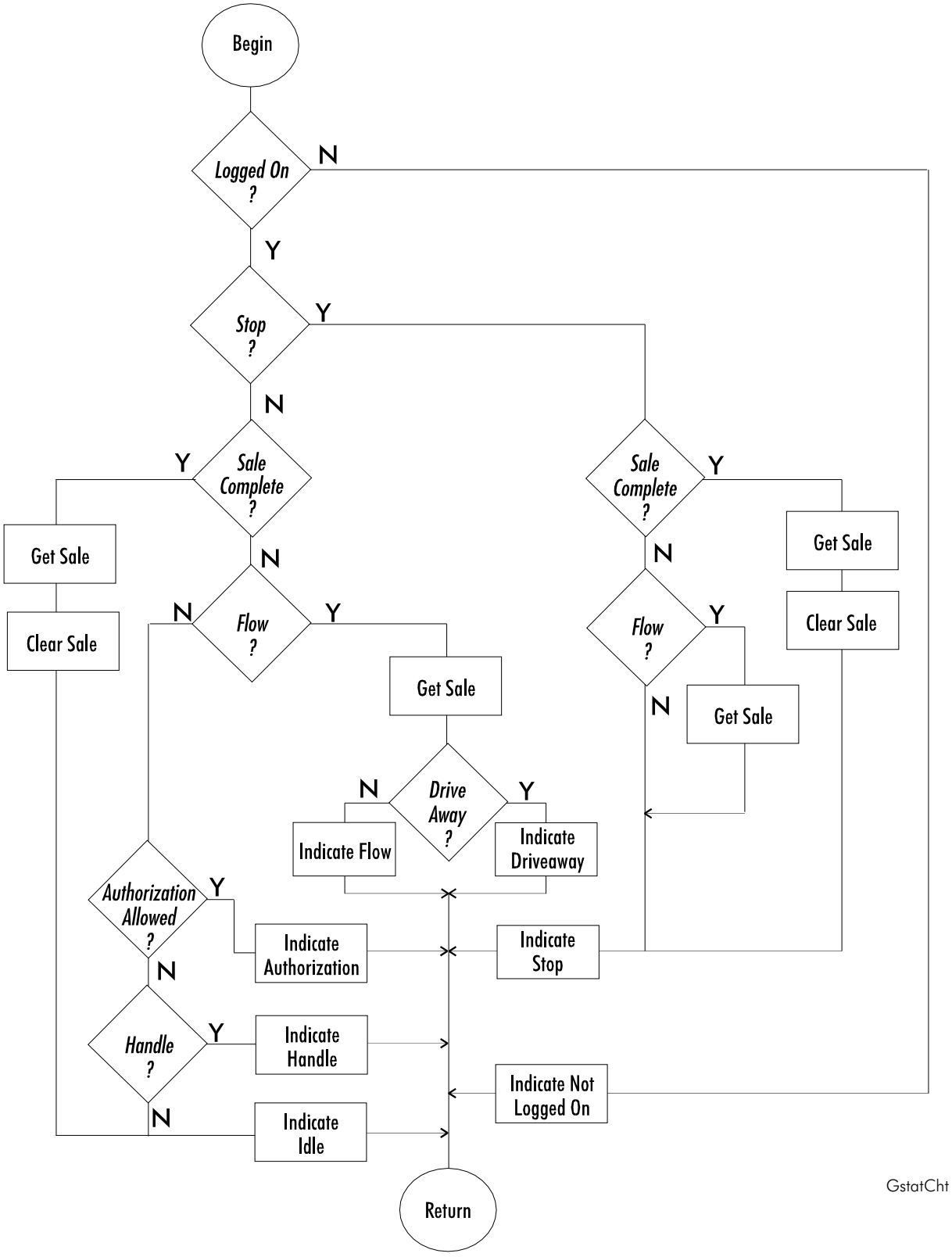
- Price information must be sent to a dispenser position to initiate communication from the controller to the dispenser. Price information indicates to the controller what style dispenser exists — single product dispenser (SPD) or multiple product dispenser (MPD). This information must match the dispenser configuration to function correctly. Pricing should never be sent for a dispenser that does not exist. Doing so will significantly slow the system operation.

- Always collect errors that the controller reports using the error command. A log file containing all reported errors should be maintained. The error log file is especially helpful in diagnosing problems during code development and site installation. This is an automatic function of the FuelDirect server for Windows-based systems.

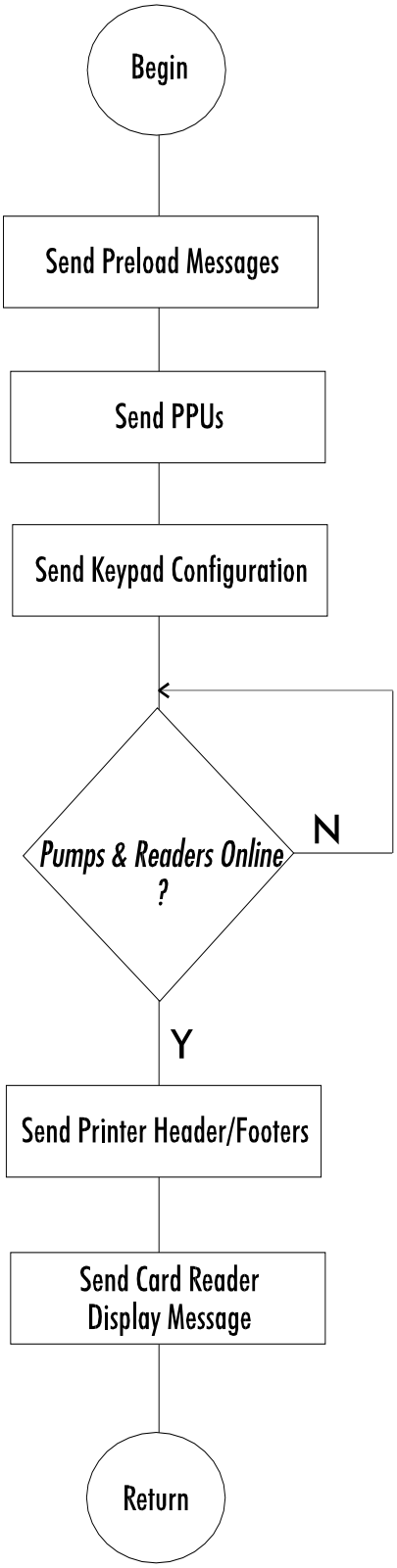
- POS architecture should service dispenser/reader operations in a timely fashion. The customer at the dispenser is not willing to wait an eternity for dispenser authorization. And to the fueling customer, three seconds is an eternity!

PIE has made every attempt to design its FuelDirect to provide seamless operation from one dispenser/reader brand to another. Still, there are some dispenser/reader brand-specific issues that the developer needs to keep in mind. For instance, a few dispenser models are unable to return polled totals. Refer to the Dispenser/Reader Specific Information listing at the end of this Dispenser Development Guide.

Dispenser Status Interrogation Flow Chart



Example: Start-up Flow Chart for Dispenser/Reader Operation



ExFloCht.wpg

This flow chart represents the general process that takes place during initialization.

FUELDIRECT — OLE2 SERVER
INTRODUCTION

This section assumes the user is well-versed in OLE2 and associated Windows programming techniques

OLE2 is the abbreviation for "object linking and embedding." A program that implements OLE2 can communicate data to other Windows applications that support OLE2. The FuelDirect program supplied by PIE to run on the POS is an OLE2 server application. It provides the POS application a fast and simple means of presenting commands and retrieving information about the dispenser and card readers.

OLE2 automation is a standard approach for accessing an object through a defined set of object interfaces. It is a standard way of making an object public to other applications. An application that exposes its objects is called an OLE2 automation server. Additional information on OLE2 may be obtained from various books and web sites. The Microsoft web site (microsoft.com) is an excellent source of information on OLE2.

Below is the process used to implement OLE2 automation with PIE's FuelDirect — OLE2 Server.

- Install the server using setup CD provided by PIE.
- Create object link to FuelDirect — OLE2 application.
- Reference method provided by FuelDirect — OLE2 Server with appropriate data.

To install the FuelDirect — OLE2 Server, run setup on the PI Application install CD. This process installs FuelDirect — OLE2 to a subdirectory and edits the Windows system registry. The registry is a Windows database which contains pointers to all OLE2 services available on your machine.

PIE's FuelDirect — OLE2 Server provides methods of controlling dispensers and card readers by exposing objects for automation. To use these methods with a language such as Visual Basic, follow the general procedure for automation as shown below.

Declare an object variable

Dim <object> as object

Create the object

Set <object> = CreateObject ("FuelDirect.Application")

Reference method in object

<object>.method = (data)

As an example, to reference the authorize method <object>.authorize=<data>

See description of data formatting below.

Next is an example using Visual Basic to Declare and Create an object, then reference one of the methods the object provides to the client application.

```
Under General Declarations:
        Public FDS as Object
        Dim OLEXfer as Variant

Private Sub Form_Load()
   Set FDS = CreateObject("FuelDirect.Application")
End Sub

Private Sub Authorize_Click()
   Dim AuthData$

   AuthData$ = "FuelingPosition=01" + Chr(13) + Chr(10)
   AuthData$ = AuthData$ + "Hose=0" + Chr(13) + Chr(10)
   AuthData$ = AuthData$ + "MOP=F" + Chr(13) + Chr(10)
   AuthData$ = AuthData$ + "Limit=999999" + Chr(13) + Chr(10)

   OLEXfer = AuthData$
  FDS.Authorize = OLExfer
End Sub
```

Data Format for FuelDirect — OLE2 Server

When referencing data to the FuelDirect — OLE2 Server, the CF_TEXT format is used in all methods, with the exception of the Packet methods. The Packet methods use the CF_DISPTEXT format.

CF_TEXT

Data is an array of text characters. Each line ends with a carriage return/linefeed (CR/LF) combination. A null character signals the end of the data.

CF_DSPTEXT

The data is a representation in text of a private data format. This data is displayed in text format in lieu of the privately formatted data.

Essentially, using a method such as our authorize method, data would be referenced as follows *(this adheres to our CF_TEXT format)*:

<object>Authorize=<datafield1>CR/LF<datafield2>CR/LF<datafield3>CR/LF<datafield4>CR/LF/NULL

Data Types Used by FuelDirect — OLE2 Server

The FuelDirect — OLE2 uses the following data-passing conventions throughout all its methods:

HRESULT <method> (VARIANT)
VARIANT <method> (VARIANT)

A brief explanation of these two data types follows. Reference books and web sites that address Windows programming techniques and OLE2 contain more advanced information on this subject. On the internet a search for HRESULT and VARIANT will be helpful for obtaining detailed descriptions of data types. The web site and reference listed below are excellent sources:

Web site www.microsoft.com

Publication **OLE Automation Programmer's Reference**
Microsoft Press ISBN 1-55615-851-3

HRESULT is the data type of the return value of all FuelDirect — OLE2 methods which do not return operation data. FuelDirect — OLE2 returns two values to report the success or failure of an operation.

Return Value	FuelDirect — OLE2	Description
0x0000 0000L	S_OK	Value of zero indicates operation success
0x0000 0001L	S_FALSE	Value indicates operation failure

VARIANT is a data type that allows for passing variable data types. Integers, strings, bools and errors are passed to/from FuelDirect — OLE2 using this data typing method. The advantage of using VARIANTS is that it provides a common data typing method used by most object-oriented languages. Due to the many different types that VARIANT supports, it is strongly recommended that the references listed above be researched.

Parameters are passed to the FuelDirect — OLE2 by one of the following:

VT_BSTR
VT_12
VT_BOOL

FuelDirect — OLE2 returns two types of VARIANT data:

VT_BSTR
VT_ERROR

The application should test the type of VARIANT return [DATA.VT] to determine whether it is string data or an error.

See the following pages for all methods provided by FuelDirect — OLE2 for dispenser/reader automation.

FUELDIRECT — OLE2 SERVER
FEATURES

Progressive International's FuelDirect — OLE2 Server provides the user with many unique dispenser management capabilities.

Provides both dispenser and reader control

Manages controller configuration:

- Sends ppu to dispensers
- Sends keyboard configuration to readers
- All dispenser brands are included (dispensers and readers)

Updates current dispenser/reader information continuously

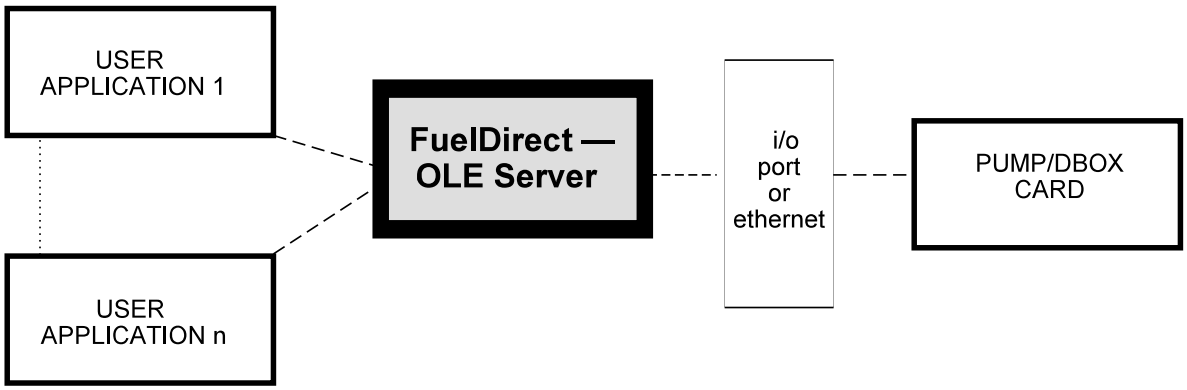
Manages the error log

Provides debugging capability by implementing these on-screen features:

- Log file
- Status icons as quick visual indicators
- Diagnostic window with numerous features

FUELDIRECT — OLE2 SERVER
THEORY AND OPERATION

The FuelDirect — OLE Server from Progressive International Electronics provides an easy to use interface to POS applications. This is a stand-alone application that is launched in SW_Hide mode and can be made visible through an interface method. A major feature of this application is that multiple POS applications may interface with the FuelDirect — OLE Server simultaneously.



FuelDirect supports Windows OLE Automation.

FUELDIRECT — OLE2 SERVER
INSTALLATION

Before installing the FuelDirect — OLE2 Server, make sure no other Windows applications are running.

- Insert CD in system.
- From the Start menu, run FuelDriect install program.
- When prompted, enter the directory location where you would like the program to be installed. The default (and suggested) path is c:\FuelDirect.

The FuelDirect — OLE2 Server is now installed.

Title Bar

FILE	PORT SELECT	DIAGNOSTICS	HELP
EXIT Exit FuelDirect — OLE2 program.	Select port #, dispenser/card brand, ethernet address. Enable/disable ATC.	Open Diagnostics window. ? will display Help menu and functions throughout Diagnostics window.	REVISIONS Display revisions file ABOUT Version(s) and release date(s) of FuelDirect — OLE2 software.

Diagnostics Window Tool Bar

FILE	CAPTURE		
EXIT Close Diagnostics window.	START Start a data capture of all data sent to the Diagnostics window and send it to the selected file. STOP Stop a data capture.		

Main Window

Indicates the current state of dispenser and reader queues.

Complete dispenser information for any specific dispenser may be viewed by clicking on that dispenser number.

Depending upon the current operation, 5 to 8 icons representing reader queues are displayed at the bottom of the screen.

An X through an icon indicates that the queue is empty.

Files

Files necessary for FuelDirect operation can be found in the FuelDirect subdirectory.

xx = fueling position

dispenserxx.ini	contains dispenser initialization and running data
error.log	contains error entries
portconfig.ini	brand and port information

Note: To run FuelDirect, pccole.tbl file must be in FuelDirect directory.

SystemVersions

Request the FuelDirect — OLE2 Server software versions

Purpose of SystemVersions Method

Request the software versions.

Interface Function Definition

VARIANT SystemVersions (void);

Syntax for SystemVersions

<data>=<OLE2object>.SystemVersions

SystemVersions Example

- Display current sections running

Method:
SystemVersions
Method Data:
System=FuelDirect 1.00 Section 1=Generic Dispenser Section 2=Generic Card Section 3=No Brand Section 4=No Brand Dongle Status=Valid Dongle

WindowControl

Control the ability to hide or show the FuelDirect — OLE2 window

Purpose of WindowControl Method

Control the ability to hide or show the FuelDirect — OLE2 Server window.

Interface Function Definition

HRESULT WindowControl (VARIANT); [VARIANT type: VSTR]

Syntax for WindowControl

<OLE2object>.WindowControl=<data>

Description of <data>

<data>	*(Description of data)*
Visible	*(Shows the FuelDirect — OLE2 Server)*
Hide	*(Hides FuelDirect — OLE2 Server from user)*

WindowControl Example

Sample Configuration:

- Hide FuelDirect — OLE2 Server from user

Method:
WindowControl
Method Data:
Hide

ReadErrorQue

Read error queue

Purpose of ReadErrorQue Method

Read error queue.

Interface Function Definition

VARIANT ReadErrorQue (void);

Syntax for ReadErrorQue

(data)=<OLE2object>.ReadErrorQue

Description of (data)

Data returned will be a string containing error information in the following format:

Device # Error Code: XXXX Error Message: "string"

"P" indicates dispenser
"C" indicates card
"String" is any additional data the server can provide to debug the error source

Table of Errors

SYSTEM-WIDE ERRORS	
1	SYSTEM_INVALID_FUNC
2	SYSTEM_INVALID_DATA
3	SYSTEM_INVALID_HANDLE
4	SYSTEM_RESOURCE_NOT_AVAILABLE
I/O ERRORS	
500	SYSTEM_NO_IO_DATA
501	SYSTEM_SERIAL_CHECK_DIGIT_ERROR
504	SYSTEM_SERIAL_COM_ERROR
507	SYSTEM_SERIAL_COM_BUSY
509	SYSTEM_SERIAL_ALL_COM_ERROR
DISPENSER ERRORS	
600	SYSTEM_DISPENSER_INVALID_TYPE
601	SYSTEM_DISPENSER_MULTI_GRADE_ERROR
602	SYSTEM_DISPENSER_SALE_DATA_ERROR
603	SYSTEM_DISPENSER_PPU_ERROR
604	SYSTEM_DISPENSER_OVERRUN_ERROR
605	SYSTEM_DISPENSER_STATUS_ERROR
606	SYSTEM_DISPENSER_NEW_PRESET_ERROR
607	SYSTEM_DISPENSER_NOT_FOUND
608	SYSTEM_INVALID_HOSE
609	SYSTEM_DISPENSER_NOT_CONFIGURED
610	SYSTEM_DISPENSER_BUSY_TRY_LATER
CARD ERRORS	
650	SYSTEM_CARD_INVALID_TYPE
602	SYSTEM_CARD_STATUS_ERROR
603	SYSTEM_CARD_NOT_FOUND
604	SYSTEM_CARD_NOT_CONFIGURED
605	SYSTEM_CARD_NO_DATA
606	SYSTEM_CARD_ENTRY_ADJUST_ERROR
607	SYSTEM_CARD_OVERFLOW_ERROR
608	SYSTEM_CARD_INVALID_DEVICE
PRINTER ERRORS	
800	SYSTEM_PRINTER_NOT_SELECTED
801	SYSTEM_PRINTER_PAPER_OUT
802	SYSTEM_PRINTER_NOT_READY
803	SYSTEM_PRINTER_BUSY_ERROR
GENERAL ERRORS	
1000	SYSTEM_GENERAL_ERROR
1001	SYSTEM_GENERAL_DIAG_ERROR

ReadErrorQue Example 1

Sample Configuration:

- Reader 10
- com error

Method:
ReadErrorQue
Method Data:
C#: 10 EC: 0504 EM: Poll Reader

ReadErrorQue Example 2

Sample Configuration:

- Dispenser 5
- Preset overrun

Method:
ReadErrorQue
Method Data:
P#: 05 EC: 0604 EM: Dispenser Sale Info

ClearErrorQue

Clear error queue

Purpose of ClearErrorQue Method

Clear the top entry in error queue.

Interface Function Definition

HRESULT ClearErrorQue (void);

Syntax for ClearErrorQue

<OLE2object>.ClearErrorQue

ClearErrorQue Example

Sample Configuration:

- Clear top entry in queue

Method:
ClearErrorQue
Method Data:
No data returned

FuelingPositionInformation

Request fueling position information

Purpose of FuelingPositionInformation Method

Retrieve all fueling position data:

- Current status
- Current sale information
- All hose information

The data may be retrieved during and at the end of a sale. After information is retrieved at the end of a sale, the sale should then be paid out, using the PayOutSale method..

Interface Function Definition

$VARIANT_1$	FuelingPositionInformation	($VARIANT_2$);	[$VARIANT_1$ type: BSTR] [$VARIANT_2$ type: VT_I2]

Syntax for FuelingPositionInformation

(returned data)=<OLE2object>.FuelingPositionInformation (passed data)

Description of (passed data)

XX	*(01 to 64)*

Description of <returned data>

Current Sale Information

FuelPosition=XX	*(01 to 64)*	
Status=XXXXXXXXXXXXXXXX	*(16 status bytes) — see description at end of this section*	
HoseX	*(Current Hose Number)*	
SaleMOP=X	*(1=Tier1; 2=Tier2)*	
SaleDollar=XXXXXXXXXX	*($12.34 = 0000001234)*	
SaleVolume=XXXXXXXXXX	*(1.234 = 0000001234)*	
TotalsType=X	*(0 = not available;*	
	2 = Money Totals Only;	
	3 = Tier1 & Tier2 Totals)	

Hoses 1-8

Hose1		
Tier1=XXXXXX	*(Tier 1 Price — PPU)*	*ex. — 1.119 = 001119*
Tier 2=XXXXXX	*(Tier 2 Price — PPU)*	*ex. — 1.119 = 001119*
Volume=XXXXXXXXXX	*(Total Volume)*	*ex. — 1.234 = 0000001234*
Tier1Dollar=XXXXXXXXXX	*(Total Dollar Tier 1)*	*ex. — $12.34 = 0000001234*
Tier2Dollar=XXXXXXXXXX	*(Total Dollar Tier 2)*	*ex. — $12.34 = 0000001234*
Blend=XXX	*(0 to 100%)*	*ex. — 100% = 100*
•		
•		
•		
Hose8		
Tier1=XXXXXX	*(Tier 1 Price — PPU)*	*ex. — 1.129 = 001109*
Tier 2=XXXXXX	*(Tier 2 Price — PPU)*	*ex. — 1.029 = 001009*
Volume=XXXXXXXXXX	*(Total Volume)*	*ex. — 1.234 = 0000001234*
Tier1Dollar=XXXXXXXXXX	*(Total Dollar Tier 1)*	*ex. — $12.34 = 0000001234*
Tier2Dollar=XXXXXXXXXX	*(Total Dollar Tier 2)*	*ex. — $12.34 = 0000001234*
Blend=XXX	*(0 to 100%)*	*ex. — 0% = 000*

High/Low Feedstock (only applicable is dispenser is a blender)

HighFeedStock=XXXXXXXXXX	*(Total HFS)*	*ex. — 1.234 = 0000001234*
LowFeedStock=XXXXXXXXXX	*(Total LFS)*	*ex. — 1.234 = 0000001234*

FuelingPositionInformation Example 1

Sample Configuration:

- Fueling Position 8
- Logged On & Handle Off Hook
- Hose 1
- Tier 1 Price
- Dollar = 2.85
- Volume = 2.546
- Money Only Totals
- Tier 1 Price = 1.119
- Tier 2 Price = 1.019
- No Blend Ratio
- Total Volume = 924356.371
- Total $ Tier 1 = 2433562.29
- Total $ Tier 2 = 0

Method:
FuelingPositionInformation(08)
Method Data :
FuelPosition=08 Status=0000000000000011 SaleHose=1 SaleMOP=1 SaleDollar-0000000285 SaleVolume=0000002546 TotalType=2 Hose1 Tier1=001119 Tier2=001019 Blend=0 Volume=0924356371 Tier1Dollar=0243356229 Tier2Dollar=0

FuelingPositionInformation Example 2

Sample Configuration:

- Fueling Position 10
- 3-Hose Dispenser
- Logged On & Handle Off Hook
- Hose 2
- Tier 2 Price
- Dollar = 4.71
- Volume = 4.582
- Tier 1 & Tier 2 Dollar Totals

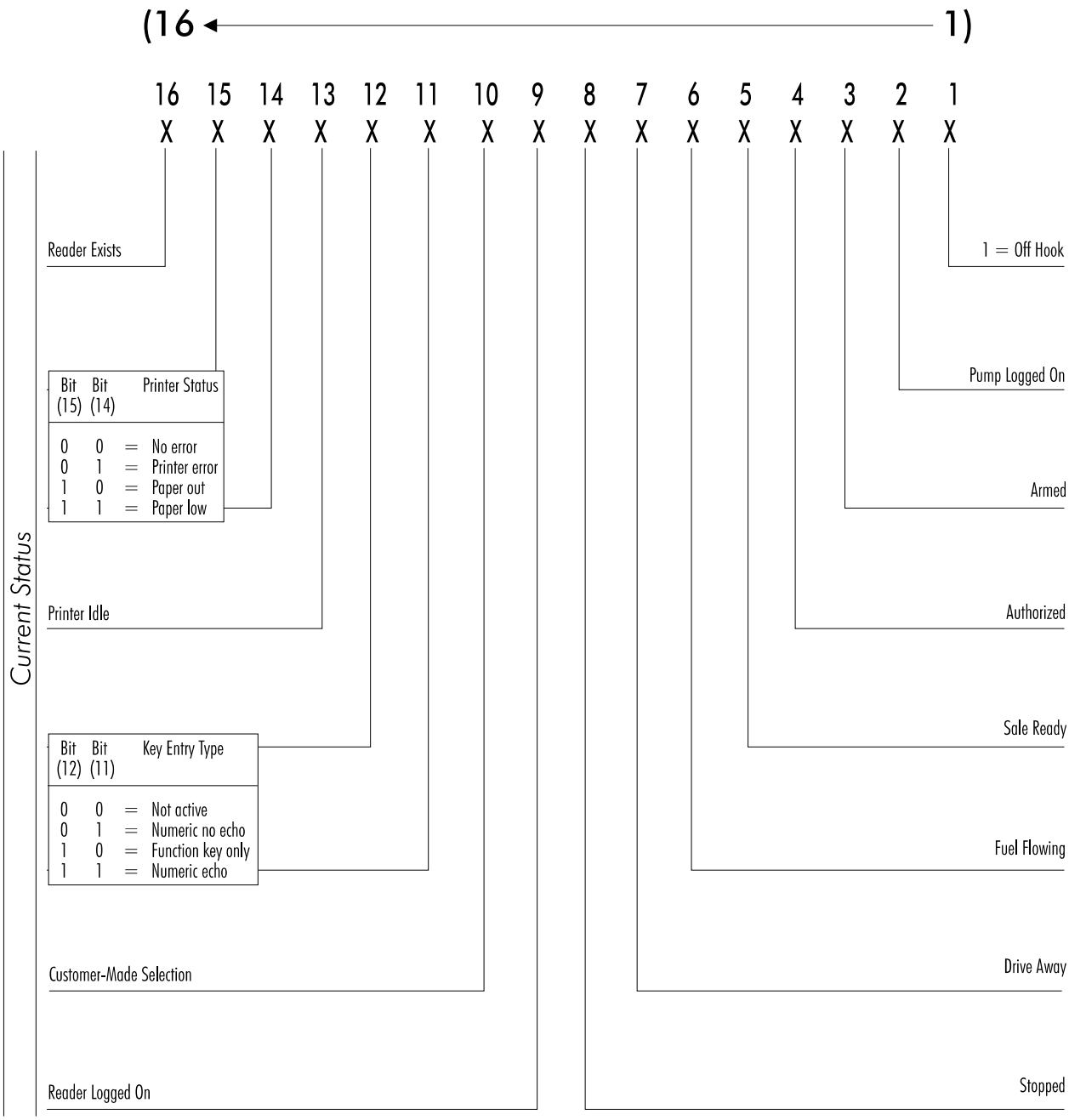
- Tier 1 Price = 1.119
- Tier 2 Price = 1.019
- Blend Ratio = 100%
- Total Volume = 924356.371
- Total $ Tier 1 = 2433562.29
- Total $ Tier 2 = 13255734.97

- Tier 1 Price = 1.129
- Tier 2 Price = 1.029
- Blend Ratio = 50%
- Total Volume = 9236.714
- Total $ Tier 1 = 29506.22
- Total $ Tier 2 = 2526073.07

- Tier 1 Price = 1.139
- Tier 2 Price = 1.039
- Blend Ratio = 0% (No Hose 1)
- Total Volume = 1025.367
- Total $ Tier 1 = 342057.68
- Total $ Tier 2 = 32010.49
- High Feed Stock = 736.789
- Low Feed Stock = 103.428

Method:
FuelingPositionInformation(10)
Method Data:
FuelPosition=10 Status=0000000000000011 SaleHose=2 SaleMOP=2 SaleDollar-0000000471 SaleVolume=0000004582 TotalType=3 Hose1 Tier1=001119 Tier2=001019 Blend=100 Volume=0924356371 Tier1Dollar=0243356229 Tier2Dollar=1325573497 Hose2 Tier1=001129 Tier2=001029 Blend=050 Volume=0009236714 Tier1Dollar=0029506229 Tier2Dollar=0252607307 Hose3 Tier1=001139 Tier2=001039 Blend=000 Volume=0001025367 Tier1Dollar=0034205768 Tier2Dollar=0003201049 HighFeedStock=0000736789 LowFeedStock=0000103428

Description of Status Bytes



Description of Status Bytes: PIE/graphics/statusbytes

Price

Send prices & blend information to fueling position

Purpose of Price Method

- Send the Tier prices (1 & 2).
- Send Blend Ratio to a fueling position and hose number.
  Blend ratio is based on the amount of Hose 1 product dispensed from the indicated hose.
- Optional Feature — Indicate the decimal places for Volume, Dollar and Price.
  Default decimal places are:
  Dollar=2
  Volume=3
  Price=3

Tier prices and blend ratios for multiple hoses may be sent at one time. Tier 1 (credit) and Tier 2 (cash) must both be sent, even if values are the same. (See examples 2 & 3.) Decimal points are only used for FuelDriect display windows. They do not set the dispenser display decimal points.

Interface Function Definition

HRESULT Price (VARIANT); [VARIANT type: BSTR]

Syntax for Price

<OLE2object>.Price=(data)

Description of (data)

FuelingPosition=XX	*(01 to 64)*
DollarDecimal=X	*(0 to 6)*
VolumeDecimal=X	*(0 to 6)*
PriceDecimal=X	*(0 to 5)*
SlowdownPoint=X	*(0 to 99)*
Hose=X	*(1 to 8)*
Tier1=XXXXXX	*(Credit 1.234 = 001234)*
Tier2=XXXXXX	*(Cash 1.234 = 001234)*
Blend=XXX	*(100% = 100)*

Price Example 1

Sample Configuration:

- Fueling Position 3
- Single Hose
- Default Decimal
- No Blend Ratio
- 2 Tiered Pricing

Method:
Price
Method Data:
FuelingPosition=03 Hose=1 Tier1=001119 Tier2=001019

Price Example 2

Sample Configuration:

- Fueling Position 6
- Three Hose
- Dollar Decimal = 1
- Volume Decimal = 2
- Price Decimal = 1
- Blend Ratios
- 2 Tiered Pricing

Method:
Price
Method Data:
FuelingPosition=06 DollarDecimal=1 VolumeDecimal=2 PriceDecimal=1 Hose=1 Tier1=001119 Tier2=001019 Blend=100 Hose=2 Tier1=001129 Tier2=001029 Blend=050 Hose=3 Tier1=001139 Tier2=001039 Blend=100

Price Example 3

Sample Configuration:

- Fueling Position 10
- Three Hose
- Dollar Decimal = 1
- Volume Decimal = 2
- Price Decimal = 1
- No Blend Ratio
- 1 Tier Pricing

Method:
Price
Method Data:
FuelingPosition=10 DollarDecimal=1 VolumeDecimal=2 PriceDecimal=1 Hose=1 Tier1=001159 Tier2=001159 Hose=2 Tier1=001199 Tier2=001199 Hose=3 Tier1=001259 Tier2=001259

Authorize

Authorize a fueling position

Purpose of Authorize Method

Activate a fueling position using the parameters of fueling position, hose number, method of payment (MOP), and maximum limit.

Interface Function Definition

HRESULT Authorize (VARIANT); [VARIANT type: BSTR]

Syntax for Authorize

<OLE2object>.AUTHORIZE=(data)

Description of (data)

FuelingPosition=XX	*(01 to 64)*
Hose=X	*(0 to 8)* or MXXXXXXXX
MOP=X	*(1 for Tier 1, 2 for Tier 2, or F for Fillup)*
Limit=XXXXXX	*($12.34 = 001234; Fillup = 999999*)

Hose is indicated as 0 for any hose.

'M' indicates multi-hose lock.

Hose=M00010100 indicates hoses 3 and 5 only are valid.

Authorize Example 1 — Fillup

Sample Configuration:

- Fueling Position 1
- Any Hose
- Fillup
- Maximum Limit

Method:
Authorize
Method Data:
FuelingPosition=01 Hose=0 MOP=F Limit=999999

Authorize Example 2 — Preset

Sample Configuration:

- Fueling Position 10
- Only Hose 2
- Tier 1
- $12.34

Method:
Authorize
Method Data:
FuelingPosition=10 Hose=2 MOP=1 Limit=001234

PayOutSale

Pay out sale on a fueling position

Purpose of PayOutSale Method

Reset the sale ready flag for a fueling position.
A fueling position cannot be authorized if this flag is set.

Interface Function Definition

HRESULT PayOutSale (VARIANT); [VARIANT type: VT_I2]

Syntax for PayOutSale

<OLE2object>.PayOutSale=(data)

Description of (data)

(data) *(Description of data — fueling positions 01 to 64)*

PayOutSale Example

Sample Configuration:
- Fueling Position 25

Method:
PayOutSale
Method Data:
25

Stop

Stop a fueling position

Purpose of Stop Method

Stop a fueling position from dispensing fuel.
To continue dispensing fuel, a issue a Resume.

Interface Function Definition

HRESULT Stop (VARIANT); [VARIANT type: VT_I2]

Syntax for Stop

<OLE2object>.Stop=(data)

Description of (data)

(data) *(Description of data — fueling positions 00 to 64)*

00 is used for an emergency stop — to stop all fueling positions.

Stop Example

Sample Configuration:

- Stop Fueling Position 25

Method:
Stop
Method Data:
25

Resume

Resume a fueling position

Purpose of Resume Method

Resume a stopped fueling position and allow fuel to flow.

Interface Function Definition

HRESULT Resume (VARIANT); [VARIANT type: VT_I2]

Syntax for Resume

<OLE2object>.Resume=(data)

Description of (data)

(data) *(Description of data — fueling positions 00 to 64)*

00 is used to resume all fueling positions.

Resume Example

Sample Configuration:

- Resume Fueling Position 8

Method:
Resume
Method Data:
08

Deauthorize

Deauthorize a fueling position

Purpose of Deauthorize Method

Terminate a sale on a fueling position.
The fueling position will either go to an Idle or a Sale Ready status. The operation may take some time. Customer may need to hang up dispenser *handle.*

Interface Function Definition

HRESULT Deauthorize (VARIANT); [VARIANT type: VT_I2]

Syntax for Deauthorize

<OLE2object>.Deauthorize=(data)

Description of (data)

(data) *(Description of data — fueling positions 01 to 64)*

Deauthorize Example

Sample Configuration:
- Deauthorize Fueling Position 31

Method:
Deauthorize
Method Data:
31

DispenserInformation

Retrieve a string containing specific dispenser information

Purpose of DispenserInformation Method

Retrieve a string containing specific dispenser information.
This information will vary in format and information. It is intended as a diagnostic tool.

Interface Function Definition

VARIANT1 = DispenserInformation (VARIANT2) (Variant1= BSTR/Variant2=VT_12)

Syntax for Deauthorize

(returned data)=<OLE2object>.DispenserInformation (Passed Data)

Description of (returned data)

FuelPosition=XX	*(01 to 64)*
Information=<text string>	*(Dispenser information)*

Description of (passed data)

FuelPosition=XX	*(01 to 64)*

DispenserInformation Example

Sample Configuration:

- Fueling Position 8
- Generic Dispenser

Method:
DispenserInformation(08)
Method Data:
FuelPosition=08 Information=Generic Dispenser PPU Displays=2 PPU displays Dispenser Type=MPD Blender=NO

KeyPadConfiguration

Set up keypad layout

Purpose of KeyPadConfiguration Method

Used to set up the keypad layout for the dispenser's card reader.
Defined keys are shown below. Remaining upper case characters are user defined. All keys must be upper case except for the "e" (end of string),"@" (unused position), and the four extended set keys used with some large keypads. The extended set keys are "a,""b," " c" and "d."

Interface Function Definition

HRESULT KeyPadConfiguration (VARIANT); [VARIANT type: BSTR]

Syntax for KeyPadConfiguration

<OLE2object>.KeyPadConfiguration=(data)

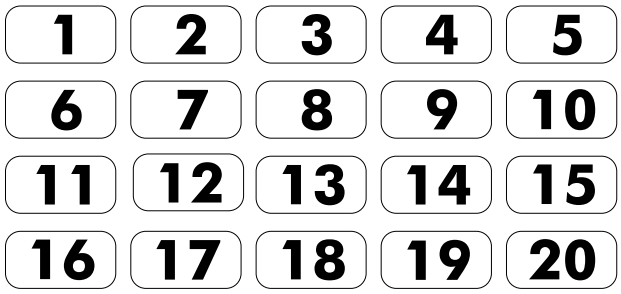
Description of (data)

ReaderPosition=XX	*(01 to 64)*
KeyData=string	*(0 to 40 keys)*

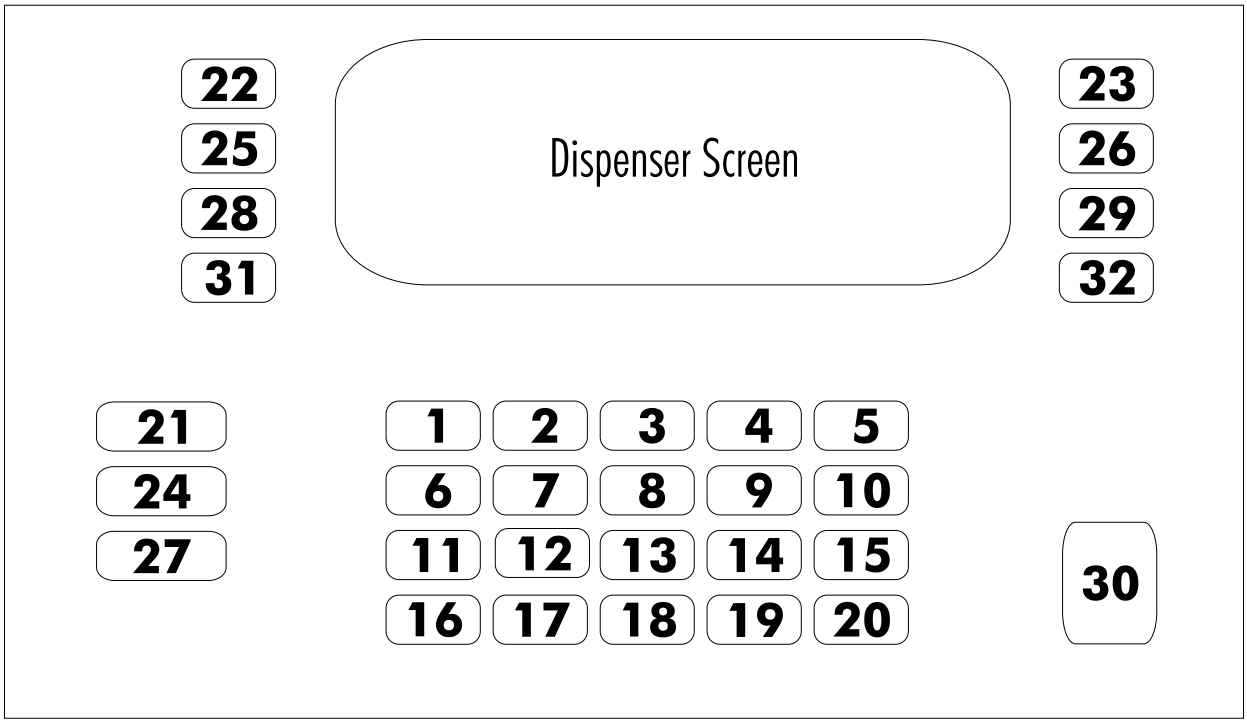
Description of Defined Keys

A = Cash	N = No
B = Back Space	R = Credit
C = Cancel	S = Start
D = Debit	Y = Yes
E = Enter	@ = Unused Position
H = Help	e = End of Key String
L = Clear	

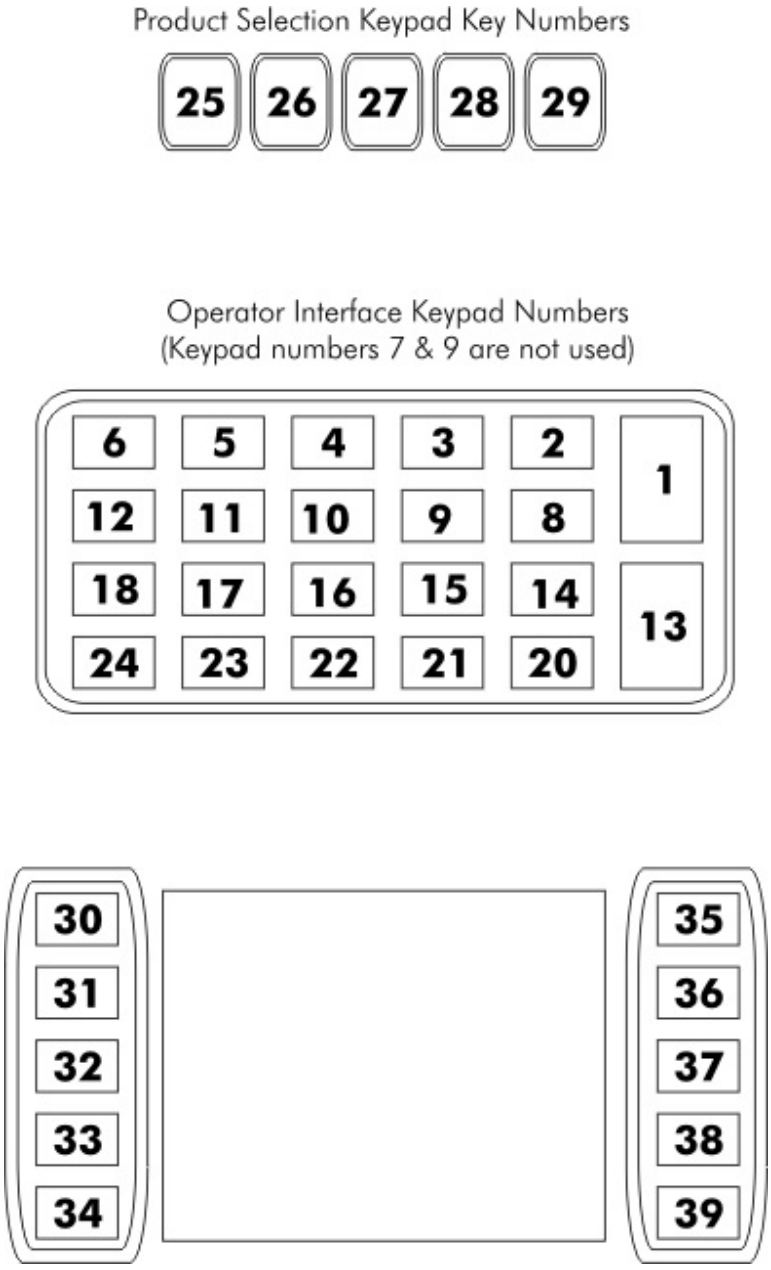
Card Reader Key Position Layouts



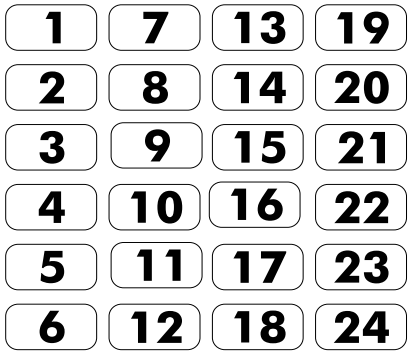
Layout: Gilbarco CRIND



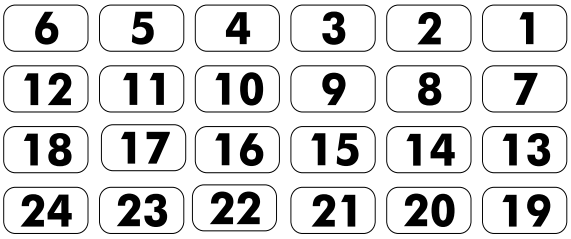
Layout: Gilbarco InfoScreen



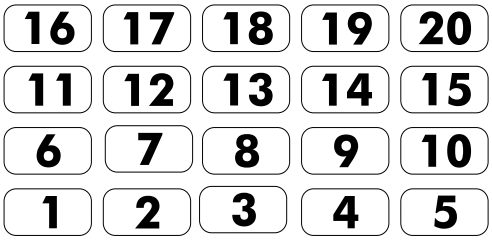
Layout: Tokheim Premier C Series In-Site



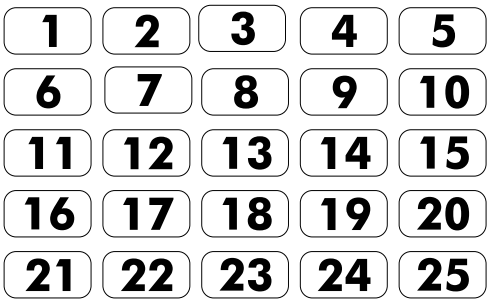
Layout: Tokheim DPT



Layout: Tokheim Premier DPT



Layout: Wayne



Layout: Schlumberger SAM

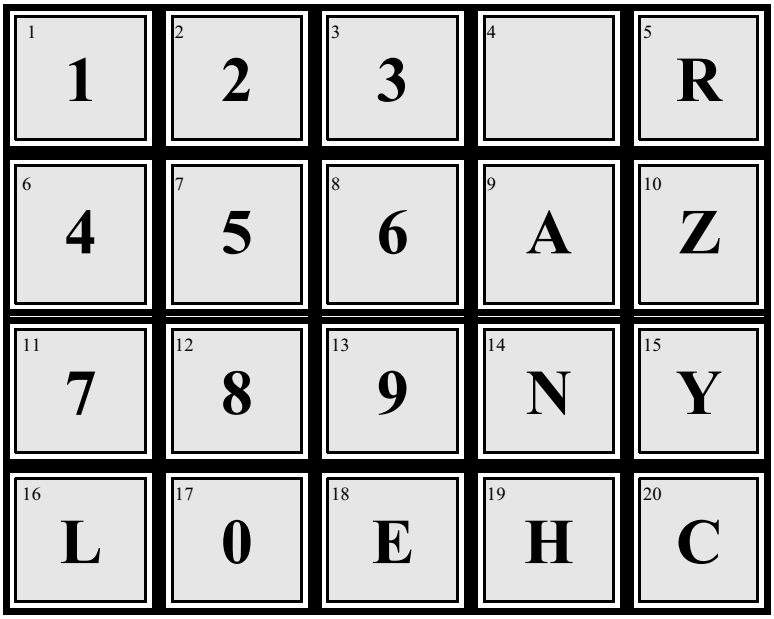
KeyPadConfiguration Example

Sample Configuration:

- Reader 1
- 20 Key Layout

Method:
KeyPadConfiguration
Method Data:
ReaderPosition=01 KeyData=123@R456AZ789NYL0EHCe

Keypad Configuration Sample



Small numbers in the upper left-hand corner of each key indicate the key position in this sample configuration of a Gilbarco CRIND™. (See Card Reader Key Position, Layout: Gilbarco CRIND.)

KeyEntryType

Enable the key pad in the mode selected

Purpose of KeyEntryType Method

Enable the key pad in the mode selected.

Interface Function Definition

HRESULT KeyEntryType (VARIANT); [VARIANT type: BSTR]

Syntax for KeyEntryType

<OLE2object>.KeyEntryType=(data)

Description of (data)

ReaderPosition=XX	*(01 to 64)*
EntryMode=string	*(AnyKey, NumericWithEcho, NumericWithoutEcho, DES encryption)*

Description of Entry Mode String

AnyKey	*All keys — function and numeric*
NumericWithEcho	*Numeric entry with echo on*
NumericWithoutEcho	*Numeric entry without echo on*
DESEncryption	*DES encryption active — assumed numeric without echo*
NumericWithEchoNoCursor	*Numeric entry with echo on and no cursor*
NumericWithoutEchoNoCursor	*Numeric entry without echo on and no cursor*
DESEncryptionNoCursor	*DES encryption active with no cursor*

KeyEntryType Example

Sample Configuration:

- Reader 2
- Numeric with echo on

Method
KeyEntryType
Method
ReaderPosition=02 EntryMode=NumericWithEcho

DisplayData

Send display information

Purpose of DisplayData Method

Send a display to the card reader in the dispenser.

Multiple lines may be sent, depending on the display type used on the reader.

A maximum of 80 characters allowed.

If reader position is set at 00, the display message will be sent to all readers that are logged on.

Scroll option — If first character is 0x13 and the display can support scrolling, the display will scroll message.

Interface Function Definition

HRESULT DisplayData (VARIANT); [VARIANT type: BSTR]

Syntax for DisplayData

<OLE2object>.DisplayData=(data)

Description of (data)

ReaderPosition=XX	*(00 to 64)*
Message=string	*(1 to 80 characters)*

Graphic Screen Control

Special Code Sequences

Graphics Message:

Must be first sequence in display message. Indicates graphics controls.

ESC G

Clear Screen:

Clears screen and sets background color. Used at start of message only.

ESC 0x1a X X X

X X X Screen Color

001 — Black	009 — Dark Gray
002 — Dark Blue	010 — Bright Blue
003 — Dark Green	011 — Bright Green
004 — Dark Cyan	012 — Bright Cyan
005 — Dark Red	013 — Bright Red
006 — Font Background	014 — Font Character Color
007 — Rust	015 — Yellow
008 — Light Gray	016 — White

All messages are displayed with the "Bright Blue" background. The Clear Screen should use this color.

Scroll Line(s):

Scrolls selected line(s). Can be used on more than one line. Must be at the beginning of the line.

ESC 0x13

Cursor Control:

Sets a cursor position in pixels.

ESC = xxxyyy

xxx = Column position in pixels
yyy - Row position in pixels

Note:

Mono Screen = 320 x 240
Color Screen = 709 x 471

Video Control:

Turns the video on and off.

ESC V x

x = 0 for OFF
x = 1 for ON

- *Background color should be kept as default. Otherwise, text-only portion is displayed with each character in a box. The background color of the box is the previous color and foreground is white.*

- *For color graphic display, scrolling displays the first line only. The second line is displayed in the second line and is centered on the second line, but is not scrolled.*

- *For monochrome graphic display, scrolling does not work. The first line is centered in the first line of the display and the second line is always below the first line, but is not necessarily centered. If not centered, it usually starts from the edge of the screen.*

- *For color graphic display, the clear screen sequence has to be sent after the enable graphics sequence. If not, the previous display gets overwritten and may be seen below the new display message. For this reason, a background color must be specified. Make sure that the specified color is the default color for the display. Otherwise, the problem mentioned in the first note will be very obvious.*

DisplayData Example 1 – Text-Only Display

Sample Configuration:

- Reader 2
- 4-line display with one line left blank

Method:
DisplayData
Method Data:
ReaderPosition=02 Message=This is Line Number 1 This is Line Number 2 *(blank line)* This is Line Number 4

A line with no text only requires the CR/LF pair to conform to the clipboard format.

DisplayData Example 2 – Graphic Display

Sample Configuration:

- Reader 3
- Graphics mode
- Screen cleared
- Cursor position:
  - x=100
  - y=100
- "HELLO" displayed

Method:
DisplayData
Method Data:
ReaderPosition=03 Message=<ESC>G<ESC><0_x1A>002<ESC>=100100HELLO

PrintOutput

Send printer information to dispenser's card reader

Purpose of PrintOutput Method

Send printer information to the card reader in the dispenser.

- Header — *Limited to 240 characters. If reader position is set at 00, the display message will be sent to all readers that are logged on.*
- Footer — *Limited to 240 characters. If reader position is set at 00, the display message will be sent to all readers that are logged on. Footer must contain enough CR/LF sequences to eject receipt from printer.*
- Receipt — *Limited to 700 characters. Multiple lines may be used.*

Interface Function Definition

HRESULT PrintOutput (VARIANT); [VARIANT type: BSTR]

Syntax for PrintOutput

<OLE2object>.PrintOutput=(data)

Description of (data)

ReaderPosition=XX	*(00 to 64)*
Job=string	*(Header, Footer, Receipt)*
Message=string	*(0 to 240 characters for Header or Footer; 0 to 700 characters for Receipt)*

PrintOutput Example #1

Sample Configuration:

- All Readers
- 4-line header with one line left blank

Method:
PrintOutput
Method Data:
ReaderPosition=00 Job=Header Message=JQ PUBLIC MINI MART 1234 ANYSTREET ANYTOWN, ANYSTATE 00000 *(blank line)*

A line with no text only requires the CR/LF pair to conform to the clipboard format.

PrintOutput Example #2

Sample Configuration:

- All Readers
- Footer

Method:
PrintOutput
Method Data:
ReaderPosition=00 Job=Footer Message=THANK YOU . . . COME AGAIN

PrintOutput Example #3

Sample Configuration:

- Reader 1
- Receipt

Method:
PrintOutput
Method Data:
ReaderPosition=01 Job=Receipt Message=SALE $19.95 VOLUME 12.55 GAL PRICE/GAL $1.059 *(blank line)*

Sample Receipt

The sample below demonstrates the receipt that will result after the steps in examples 1, 2 & 3 — for header, footer and receipt — have been implemented.

Special Code Sequences

Barcode Print Control:

Print data can have an embedded control sequence to print a barcode. The following sequence is used to print a barcode on the receipt.

ESC B XXXXXXXXXXXXX ETX

The X's represent the digits of the barcode. There must be 13 numbers in the barcode with no spaces in the sequence.

BeeperControl

Sound the beeper a specified number of times

Purpose of BeeperControl Method

Sound the beeper the number of times specified.

Interface Function Definition

HRESULT BeeperControl (VARIANT); [VARIANT type: BSTR]

Syntax for BeeperControl

<OLE2object>.BeeperControl=(data)

Description of (data)

Reader Position=XX	*(01 to 64)*
BeepCounts=XX	*(01 to 10 — number of times set to beep)*

BeeperControl Example

Sample Configuration:

- Sound Beeper 3 Times on Reader 2

Method:
BeeperControl
Method Data:
ReaderPosition=02 BeepCounts=03

Refer to documentation accompanying the reader for reader specific restrictions.

PreloadDisplayMessage

Preload display messages in the readers

Purpose of PreloadDisplayMessage Method

Preload display messages in the readers. These messages are for all readers on each bank.

Messages must be sent prior to sending the KeyPad configuration for the reader.

Interface Function Definition

HRESULT PreloadDisplayMessage (VARIANT) [VARIANT type: BSTR]

Syntax for PreloadDisplayMessage

<OLE2object>.PreloadDisplayMessage=(data)

Description of (data)

Section=Y	(*Bank Selection:* *0=all dispenser/reader banks — all banks* *1=dispenser/reader bank 1 — 1-16* *2=dispenser/reader bank 2 — 17-32* *3=dispenser/reader bank 3 — 33-48* *4=dispenser/reader bank 4 — 49-64)*
MessageType=string	*(Card, Printer, Keypad)*
MessageNumber=X	*(See description which follows)*
Message=string	*(User defined text string)*

For default display message, Y must equal 0.

Description of Message Numbers

Printer Default Messages

1 Printing Receipt
2 Receipt Complete
3 Please Take Receipt
4 Printer Error

Card Default Messages

1 Remove Card Quickly
2 Insert Card Again
3 Invalid Card
4 One Moment Please
5 Card Inserted Wrong/Please Try Again

Keyboard Default Messages

1 Cancel Key Message
2 Invalid Key

Display Default Message

1 Message to Display at Startup

PreloadDisplayMessage Example

Sample Configuration:

- Display message to both banks

Method
PreloadDisplayMessage
Method Data:
Section=0 MessageType=Card MessageNumber=2 Message=Please Reinsert Card

QueCountQuery

Query the entries in the incoming queues

Purpose of QueCountQuery Method

Query the entries in the incoming queues — Card, Key, Cash and Packet

Interface Function Definition

VARIANT QueCountQuery (void);

Syntax for QueCountQuery

(data)=<OLE2object>.QueCountQuery

Description of (data)

CardCount=XXX	*(000 to 999 — # of Card entries in queue)*
KeyCount =XXX	*(000 to 999 — # of Key entries in queue)*
CashInCount=XXX	*(000 to 999 — # of Cash entries in queue)*
PacketInCount =XXX	*(000 to 999 — # of Packet entries in queue)*
ErrorQueCount=XXX	*(000 to 999 — # of Error entries in queue)*

QueCountQuery Example

Sample Configuration:

- 4 readers in card queue
- 1 key read in key queue

Method:
QueCountQuery
Method Data:
CardCount=004 KeyCount=001 CashInCount=000 PacketInCount=000 ErrorQueCount=000

CardQue

Retrieve the top entry from the card queue

Purpose of CardQue Method

Retrieve the data from the top entry in the card queue.

Interface Function Definition

VARIANT CardQue (void);

Syntax for CardQue

(data)=<OLE2object.>CardQue

Description of (data)

ReaderPosition=XX	*(01 to 64)*
Track1=string	*(track 1 data)*
Track2=string	*(track 2 data)*
Track3=string	*(track 3 data)*

When none of the three tracks show data, a bad read on a card reader is indicated.

CardQue Example

Sample Configuration:

- Reader Position 12
- No Data on Tracks 1 or 3
- Track 2 Data = 12345611127634567

Method:
CardQue
Method Data:
ReaderPosition=12 Track1= Track2=1234561127634567 Track3=

KeyQue

Retrieve the top entry from the key queue

Purpose of KeyQue Method

Retrieve the data from the top entry in the key queue.

Interface Function Definition

VARIANT KeyQue (void);

Syntax for KeyQue

(data)=<OLE2object>.KeyQue

Description of (data)

ReaderPosition=XX	*(01 to 64)*
KeyCount=XXX	*(# of bytes)*
Data=string	*(Key data bytes)*

KeyQue Example

Sample Configuration:

- Reader Position 24
- 5 data bytes
- Data = 1234E

Method:
KeyQue
Method Data:
ReaderPosition=24 KeyCount=005 Data=1234E

CashInQue

Retrieve the top entry from the cash queue

Purpose of CashInQue Method

Retrieve the data from the top entry in cash queue. Retrieve top entry from cash queue.

Interface Function Definition

VARIANT CashInQue (void);

Syntax for CashInQue

(data)=<OLE2object.>CashInQue

Description of (data)

ReaderPosition=XX	*(01 to 64)*
CashCount=XXX	*(Number of characters in CashData – 3 digits)*
CashData=XXXXXXXXXX	*($s returned)*

Format for (data)

$S_1S_2S_3S_4S_5S_6$[sp]CB=XXX[sp]AB=YYY

- S_1 Status 1 (controller status) – *see following Status Return Chart*
- S_2 Status 2 (controller status)
- S_3 Status 3 (controller status)
- S_4 Status 4 (controller status)
- S_5 Status 5 (acceptor status)
- S_6 Status 6 (acceptor status)
- XXX Current bill in acceptor (3 digit $ value)
- YYY Accumulated bills in acceptor since last clear (3 digit $ value)
- [sp] Space character [0x20]

A solicited cash acceptor status request always returns a CB=000

Status Return Values

All values returned as hex values (30h-46h) rep0resenting 0-9 and A-F

Controller Status Codes (bit oriented values)

S1	S2	S3	S4	
0	0	0	0	Cash acceptor disabled
x	x	x	1	Cash acceptor enabled
x	x	1	x	Solicited status
•	•	•	•	Reserved values
•	•	•	•	
[h]	[h]	[h]	[h]	

Acceptor Status Codes (numeric values)

S5	S6	
0	0	Note acceptor idle
0	1	Note escrowed
0	2	Note stacked
0	3	Note returned
0	4	Bill rejected
0	5	Acceptor jammed
0	6	Acceptor cassette full
0	7	Acceptor cassette removed
0	8	Acceptor cassette replaced
0	9	Acceptor power up
•	•	
•	•	Reserved codes
•	•	
F	E	Note status unknown**
F	F	Note acceptor error

** *If this status is returned, an additional hex value is returned [hh], showing the value the controller was given*

x represents a variable bit value

[h] represents a hexadecimal digit

It is highly recommended that an audit trail file for all cash queue related data (cash queue commands and cash queue responses) be maintained

CashInQue Example

Sample Configuration:

- Reader 4
- Byte count 19
- Cash acceptor enabled
- Bill escrowed
- Current bill is $1.00
- Accumulated bills $10.00

Method:
CashInQue
Method Data:
ReaderPosition=04 CashCount=019 CashData=111101 CB=001 AB=010

CashOutQue

Send command to reader cash acceptor

Purpose of CashOutQue Method

Send command to reader cash acceptor.

Interface Function Definition

HRESULT CashOutQue (VARIANT); [VARIANT type:BSTR]

Syntax for CashOutQue

<OLE2object>.AUTHORIZE=<data.>

Description of (data)

ReaderPosition=XX	*(01 to 64)*
CashCount=XXX	*(Number of characters in CashData – 3 digits)*
CashData=XXXXXXXXXX	*(Command — see following Description of Command)*

Description of Command

[optional data[field is used to send additional numeric data with certain commands shown below.

EE Enable cash acceptor (2 byte flag)
ED Disable cash acceptor (2 byte flag)
Z Cash acceptor status request*
S Stack bill in acceptor **
R Return bill to customer***
C Clear accumulated bills
T Set maximum bill type — use with following optional data fields:
- 001 $1 bill maximum
- 005 $5 bill maximum
- 010 $10 bill maximum
- 020 $20 bill maximum
- 050 $50 bill maximum
- 100 $100 bill maximum

0 Set bill orientation — use with following optional data fields:
- B Black side up
- G Green side up
- 2 Both directions

* Forces a cash acceptor status request queue entry. A solicited cash acceptor status request always returns a CB=000.

** Generates two cash acceptor status request queue entries (stacked and idle).

*** Generates two cash acceptor status reques queue entries (returned and idle).

CashOutQue Example 1 — Enable Cash Acceptor

Sample Configuration:

- Reader Position 5
- Maximum bill is $20
- Both sides
- Enable cash acceptor

Method:
CashOutQue
Method Data:
ReaderPosition=05 CashCount=008 CashData=T020O2EE

CashOutQue Example 2 — Request Current Cash Acceptor Status

Sample Configuration:

- Reader Position 1
- Request current cash acceptor status

Method:
CashOutQue
Method Data:
ReaderPosition=01 CashCount=001 CashData=Z

CashOutQue Example 3 — Stack Bill

Sample Configuration:

- Reader Position 5
- Stacking escrowed bill

Method:
CashOutQue
Method Data:
ReaderPosition=05 CashCount=001 CashData=S

CashOutQue Example 4 — Return Bill

Sample Configuration:

- Reader Position 6
- Return escrowed bill

Method:
CashOutQue
Method Data:
ReaderPosition=06 CashCount=001 CashData=R

PacketQueInput

Retrieves data directly from the reader

Purpose of PacketQueInput Method

Retrieves data directly from the reader. *Packet data will be in the format of actual brand-specific reader protocol.*

Interface Function Definition

VARIANT PacketQueInput (void);

Syntax for PacketQueInput

(data)=<OLE2object.>PacketQueInput

PacketQueInput method does not use the standard clipboard format, CF_TEXT. Instead, the CF_DISPTEXT format is used.

Description of (data)

ReaderPosition=XX	*(01 to 64)*
PacketCount=XXX	*(Number of characters in packet count – 3 digits)*
PacketData=XXX . . . XX	*(Response from reader)*

PacketQueInput Example

Sample Configuration:

- Reader Position 6
- Brand-specific reader response

Method:
PacketQueInput
Method Data:
ReaderPosition=06 PacketCount=008 PacketData=T020O2EE

PacketQueOutput

Sends data directly to the reader

Purpose of PacketQueOutput Method

Sends data directly to the reader. *Packet data must be actual brand-specific reader protocol.*

Interface Funtion Definition

HRESULT PacketQueOutput (VARIANT) [VARIANT type: BSTR]

Syntax for PacketQueOutput

<OLE2object>.PacketQueOutput=(data)

Packet Que methods do not use the standard clipboard format, CF_TEXT. Instead, the CF_DISPTEXT format is used.

Description of (data)

ReaderPosition=XX	*(01 to 64)*
PacketCount=XXX	*(Number of characters in packet count – 3 digits)*
PacketData=XXX . . . XX	*(Response from reader)*

PacketQueOutput Example

Sample Configuration:

- Reader Position 4
- Brand-specific reader command

Method:
PacketQueOutput
Method Data:
ReaderPosition=04 PacketCount=008 PacketData=T020O2EE

ClearTopCardQue

Clears top entry in the card queue

Purpose of ClearTopCardQue Method

Clears the top entry in the card queue.

Interface Function Definition

HRESULT ClearTopCardQue (void);

Syntax for ClearTopCardQue

<OLE2object.>ClearTopCardQue

ClearTopCardQue Example

Sample Configuration:

- Clear top entry in queue

Method:
ClearTopCardQue
Method Data:
No data returned

ClearTopKeyQue

Clears top entry in the key queue

Purpose of ClearTopKeyQue Method

Clears the top entry in the key queue.

Interface Function Definition

HRESULT ClearTopKeyQue (void);

Syntax for ClearTopKeyQue

<OLE2object.>ClearTopKeyQue

ClearTopKeyQue Example

Sample Configuration:

- Clear top entry in queue

Method:
ClearTopKeyQue
Method Data:
No data returned

ClearTopCashInQue

Clear top entry in the cash queue

Purpose of ClearTopCashInQue Method

Clear the top entry in the cash queue.

Interface Function Definition

HRESULT ClearTopCashInQue (void);

Syntax for ClearTopCashInQue

<OLE2object.>ClearTopCashInQue

ClearTopCashInQue Example

Sample Configuration:

- Clear top entry in queue

Method:
ClearTopCashInQue
Method Data:
No data returned

ClearPacketEntry

Clear top entry in the packet input queue

Purpose of ClearPacketEntry Method

Clear the top entry in the packet input queue.

Interface Function Definition

HRESULT ClearPacketEntry (void);

Syntax for ClearPacketEntry

<OLE2object.>ClearPacketEntry

Packet Que methods do not use the standard clipboard format, CF_TEXT. Instead, the CF_DISPTEXT format is used.

ClearPacketEntry Example

Sample Configuration:

- Clear top entry in queue

Method:
ClearPacketEntry
Method Data:
No data returned

ReaderInformation

Retrieve a string containing specific reader information

Purpose of ReaderInformation Method

Retrieve a string containing specific reader information.
This information will vary in format and information. It is intended as a diagnostic tool.

Interface Function Definition

VARIANT1 = ReaderInformation (VARIANT2) (Variant1= BSTR/Variant2=VT_12)

Syntax for ReaderInformation

(returned data)=<OLE2object>.ReaderInformation (Passed Data)

Description of (returned data)

ReaderPosition=XX	*(01 to 64)*
Information=<text string>	*Dispenser information*

Description of (passed data)

Reader Position=XX	*(01 to 64)*

ReaderInformation Example

Sample Configuration:

- Fueling Position 2
- Generic card reader

Method:
ReaderInformation(02)
Method Data:
ReaderInformation=02 Information=Generic Card Reader Cash Acceptor=YES Display=4x20 TEXT Barcode Reader=YES

TankMonitorStartReport

Initiate action to receive a tank report

Purpose of TankMonitorStartReport Method

Initiate action to receive a tank report.

Interface Function Definition

HRESULT = TankMonitorStartReport (VARIANT) (VARIANT type VT_12)

Syntax for TankMonitorStartReport

<OLE2object>.TankMonitorStartReport=(data)

Description of (data)

TankNumber=XX *(01 to 08)*

TankMonitorStartReport Example

Sample Configuration:

- Tank 3

Method:
TankMonitorStartReport
Method Data:
03

TankMonitorReportStatus

Receive current status of tank report

Purpose of TankMonitorReportStatus Method

Receive current status of tank report.

Interface Function Definition

VARIANT TankMonitorReportStatus (void)

Syntax for TankMonitorReportStatus

(data)=<OLE2object>.TankMonitorReportStatus

Description of (data)

Tank Monitor Status = BUSY

or

Tank Monitor Status = DONE

TankMonitorReportStatus Example

Sample Configuration:

- Report completed

Method:
TankMonitorReportStatus
Method Data:
Tank Monitor Report Status=Done

TankMonitorReport

Get tank report

Purpose of TankMonitorReport Method

Get tank report.

Interface Function Definition

VARIANT TankMonitorReport (void)

Syntax for TankMonitorReport

(data)=<OLE2object>.TankMonitorReport

Description of (data)

Tank Monitor = XX	*(01 to 08)*
Inventory Report =(string)	*text string*
Alarm Report =(string)	*text string*
Leak Report =(string)	*text string*

TankMonitorReport Example

Sample Configuration:

- Tank monitor 2
- Inventory report
- Alarm report
- Leak report

String information is data returned from a TLS250-compatible tank monitor.

Method:
TankMonitorReport(02)
Method Data:
TankNumber=02 Inventory Report=(string) Alarm Report=(string) Leak Report=(string)

CarWashRequestStatus

Return the current car wash status bits

Purpose of CarWashRequestStatus Method

Return the current car wash status bits.

Interface Function Definition

VARIANT CarWashRequestStatus (void)

Syntax for CarWashRequestStatus

<data>=<OLE2object>.CarWashRequestStatus

Description of (data)

Current Car Wash Status

Customer Status=0000000000000000	Reserved
Controller Status=0000000000000000	Reserved
Car Wash Status=00000000000000X0	(Bit 2 indicates out of service if set to 1)

CarWashRequestStatus Example

Sample Configuration:

- Car wash out of service

Method:
CarWashRequestStatus
Method Data:
Customer Status=0000000000000000 Controller Status=0000000000000000 Wash Status=0000000000000010

Definitions of Car Wash Status Bits *Of the status bits, note that bit 2 is the only one currently in use. The remaining status bits are reserved for future implementation.*		
Customer Status (right to left)		
Bit 1	Key Pressed	Not Supported
Bit 2	Numeric Data Ready	Not Supported
Bit 3	Valid RFID	Not Supported
Bit 4	Valid Card	Not Supported
Bit 5	Card Retry Reached	Not Supported
Bit 6	Transaction Ready	Not Supported
Bit 7	Transaction Cancelled	Not Supported
Controller Status (right to left)		
Bit 1	Printer Error	Not Supported
Bit 2	Printer Paper Low	Not Supported
Bit 3	Bill Acceptor Error	Not Supported
Bit 4	Coin Acceptor Error	Not Supported
Bit 5	Coin Dispenser Error	Not Supported
Wash Status (right to left)		
Bit 1	System Error	Not Supported
Bit 2	Wash Out of Service	Supported
Bit 3	Wash Chemical Low	Not Supported
Bit 4	Wash Chemical Out	Not Supported
Bit 5	Wash Position Error	Not Supported
Bit 6	Wash Controller Error	Not Supported
Bit 7	Wash Armed	Not Supported

CarWashOperationStatus

Return the current status of the requested data operation

Purpose of CarWashOperationStatus Method

Return the current status of the requested data operation.

Interface Function Definition

VARIANT CarWashOperationStatus (void)

Syntax for CarWashOperationStatus

<data>=<OLE2object>.CarWashOperationStatus

Description of (data)

Current Car Wash Status=IDLE	No commands are in progress
Car Wash Status=BUSY	Command in progress (wait)
Car Wash Status=DATA READY	Data is ready to read
Car Wash Status=UNKNOWN	Unknown status (error)

CarWashOperationStatus Example

Sample Configuration:

- Last operation request is busy

Method:
CarWashOperationStatus
Method Data:
Car Wash Status=BUSY

CarWashInformationRequest

Retrieve requested information from the car wash

Purpose of CarWashInformationRequest Method

Retrieve requested information from the car wash.

Interface Function Definition

HRESULT CarWashInformationRequest (VARIANT TYPE VT_BSTR)

Syntax for CarWashInformationRequest

HRESULT = <OLE2object>.CarWashInformationRequest = <data>

Description of (data)

RequestCarWashCode — Request a code for a car wash

- ProgramNumber=XX
- PriceLevel=XX
- OptionCount=XX
- OptionNumber=XX
- PriceLevel=XX
  - •
  - •
- OptionNumber=XX
- PriceLevel=XX

Returns Car Wash Code

- CarWashCode=XXXXXXXXXX
- CodeValue=XXXXX

CancelCarWashCode — Cancel an approved code

- WashCode=XXXXXXXXXX

CarWashCodeStatus	Status of car wash code
WashCode=XXXXXXXXXX	
SetPriceDate	(Maximum of 12 items)
ProgramCount=XX	
ProgramNumber=01	
FullPrice=XXXXX	
Discount1Price=XXXXX	
Discount2Price=XXXXX	
•	
ProgramNumber=nn	(Last Program to be set)
FullPrice=XXXXX	
Discount1Price=XXXXX	
Discount2Price=XXXXX	
•	
OptionCount=XX	
OptionNumber=01	
FullPrice=XXXXX	
Discount1Price=XXXXX	
Discount2Price=XXXXX	
•	
OptionNumber=nn	(Last Option to be set)
FullPrice=XXXXX	
Discount1Price=XXXXX	
Discount2Price=XXXXX	
ReturnPriceData	Return the Level Price information
ProgramCount=XX	
ProgramNumber=01	
ProgramName=”Express”	
FullPrice=XXXXX	
Discount1Price=XXXXX	
Discount2Price=XXXXX	
•	
ProgramNumber=nn	(Last Program)
ProgramName=”Works”	
FullPrice=XXXXX	
Discount1Price=XXXXX	
Discount2Price=XXXXX	

•	
OptionCount=XX OptionNumber=01 OptionName=”Option 1" FullPrice=XXXXX Discount1Price=XXXXX Discount2Price=XXXXX	
•	
OptionNumber=nn OptionName=”Option nn” FullPrice=XXXXX Discount1Price=XXXXX Discount2Price=XXXXX	(Last Option)
ReturnReportData	Return the current Audit report (brand dependent)
ReturnTransactionData	(Not implemented)
ReturnKeyData	(Not implemented)
ReturnCardData	(Not implemented)
ReturnRFIDData	(Not implemented)

CarWashInformationRequest Examples

Sample Configuration:

- **Request Car Wash Code**

  Return data in the code for the car wash and the price of the car wash specified

Method:
CarWashInformationRequest
Method Data:
Request CarWashCode\r\n ProgramNumber=01\r\n PriceLevel=0\r\n OptionCount=02\r\n OptionNumber=01\r\n PriceLevel-0\r\n OptionNumber=03\r\n PriceLevel=1\r\n

Sample Configuration:

- **Set Program and Option Prices**

Method:
SetPriceData
Method Data:
SetPriceData\r\n ProgramCount=03\r\n ProgramNumber=01\r\n FullPrice=00300\r\n Discount1Price=00200\r\n Discount2Price=00100\r\n ProgramNumber=02\r\n FullPrice=00400\r\n Discount1Price=00300\r\n Discount2Price=00200\r\n ProgramNumber=03\r\n FullPrice=00500\r\n Discount1Price=00400\r\n Discount2Price=00300\r\n\0 OptionCount=03\r\n OptionNumber=01\r\n FullPrice=00100\r\n Discount1Price=00050\r\n Discount2Price=00045\r\n OptionNumber=02\r\n FullPrice=00040\r\n Discount1Price=00030\r\n Discount2Price=00020\r\n OptionNumber=03\r\n FullPrice=00040\r\n Discount1Price=00030\r\n Discount2Price=00025\r\n\0

CarWashReadData

Retrieve requested data from the car wash

Purpose of CarWashReadData Method

Retrieve requested data from the car wash.

Interface Function Definition

VARIANT CarWashReadData (void)

Syntax for CarWashReadData

<data>=<OLE2object>.CarWashReadData

Description of (data)

The returned data depends on the command requested

CarWashReadData Example

Method:
CarWashReadData
Method Data:
Data and format are specific to command and car wash brand

Dispenser Specific Information

A few commands are handled differently due to a particular dispenser's proprietary protocol. Note any specifics related to pump brand(s) being interfaced with PIE products.

Tokheim

Authorize Command — new limit

- Dispenser must support the AE46 command.

Gilbarco

Authorize Command — new limit

- New limit must be sent prior to flow.

Wayne-Dresser

Authorize Command — new limit

- New limit may be sent at any time prior to previous limit being reached.
- If increasing or decreasing original limit set, allow for some delay in communication. If insufficient time is allowed, error message may result or new limit may not be attained.

Schlumberger

Authorize Command — new limit

- New limit must be sent prior to flow.

Reader Specific Information

Tokheim

- Display:
  - 4 x 20 characters
  - no graphics
  - scrolling not supported
- DES encryption to be supported — *Contact PIE for release date*
- Beeper options — 1 or 3
- Reader software must be Version JP020800 or later

Gilbarco

- Display:
  - 1 x 20 characters
  - Scroll Flag $0x13_{HEX}$ *(must be first display character)*
- DES encryption supported
- Beeper options — 1 to 10
- Only Print Default preloadable messages are supported — P1 through P4
- Receipts must be a minimum of 40 lines long
- CRIND software must be version 51.1.6 or later

Wayne-Dresser

- Display:
  2 x 20 characters
  scrolling not supported
- DES encryption supported
- DES key read returned in packet with K as first character
  If DES key read is double-wide, one position must be unused
- Packet Command Flag
  Include as first character of data flag to describe to which type device the packet is being sent — card reader or MSM
  C = Card Reader
  M = MSM
  Device #01 Bank 0 MSM
  Device #17 Bank 1 MSM
  Device#33 Bank 2 MSM
  Device #49 Bank 3 MSM

- Beeper options — 1 to 10
- Only Print and Card Default preloadable messages are supported — P1-P4 and C1-C5

Schlumberger

- Display:
  2 x 16 characters
  no graphics
  scrolling supported
- DES encryption to be supported — *Contact PIE for release date*
- Beeper options — 1 to 10
- Only Card Default preloadable messages are supported — C1-C5

GENERIC DISPENSER SIMULATOR PROGRAM
INSTALLATION, RUNNING AND OPERATION

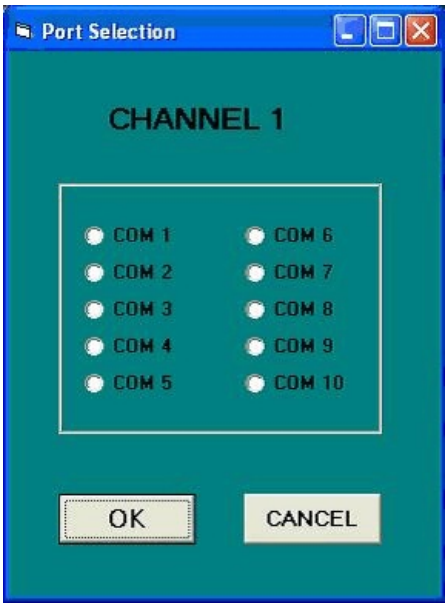
Progressive provides a generic dispenser and reader simulator which emulates the operation of a dispenser and reader. This facilitates an important step in the development process.

Install Program

Insert PI Application CD into drive. From the PC's Start menu, select Run, then Browse the CD, then Dispenser Simulator folder. Double-click on Setup.

Run Program

Double-click the PumpSim icon to run the simulator program. The Port Selection screen will appear, prompting a selection for a comp port on your PC. *See Port Selection screen below.*

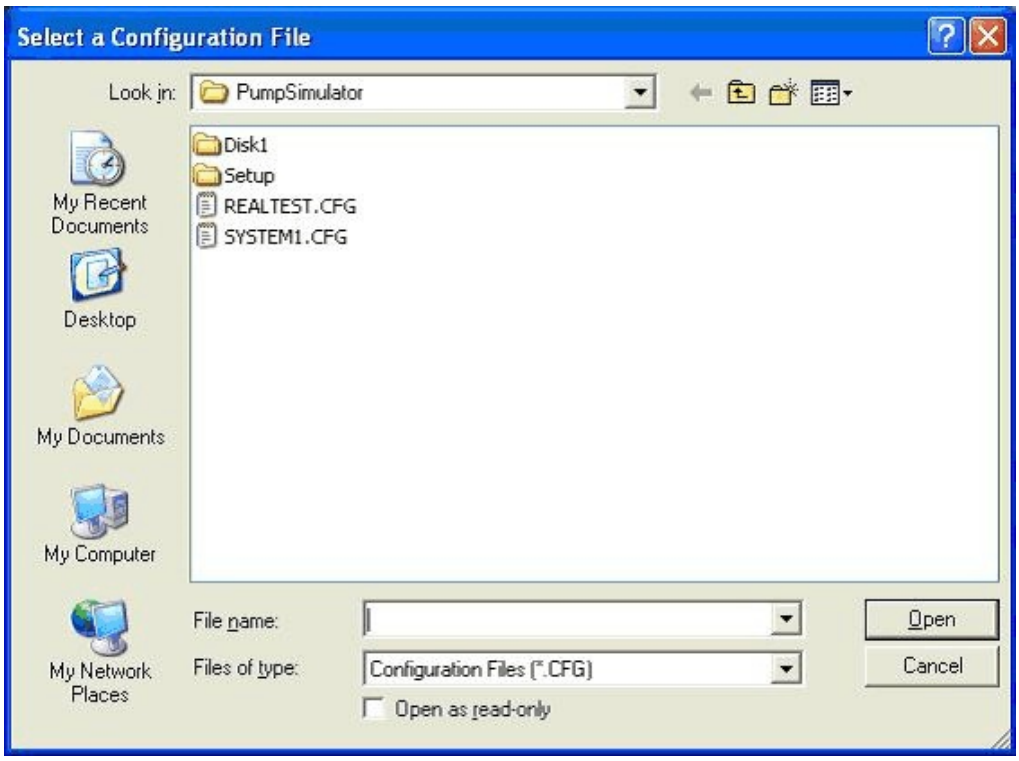


After making a port selection, the PumpSim Screen will come up and you will be prompted to select whether you wish to set up a new system configuration or use an existing configuration. *See PumpSim screen below.*

Note: It is always faster to set up a new system.

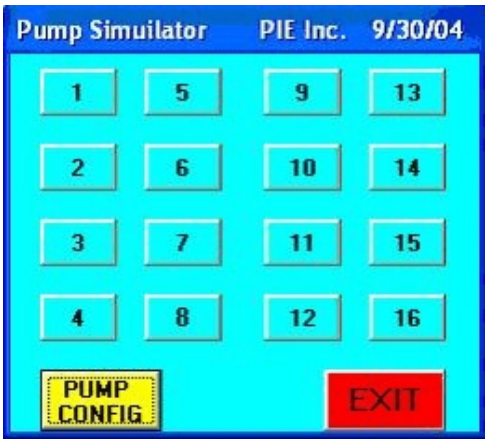
Existing System Configuration ("No" Selected from PumpSim Screen)

If "no" is selected, to use an existing system configuration, a Windows file dialog box appears. Simply locate and select the saved configuration to be run. *See Select a Configuration File screen below.*

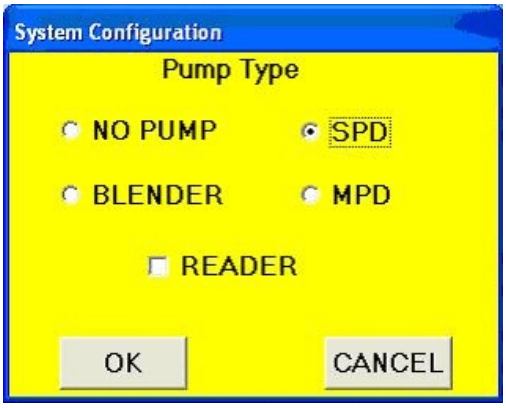


New System Configuration ("Yes" Selected from PumpSim Screen)

If "yes" is selected for a new system a screen will appear to select which dispenser/reader you wish to set up. *See Pump Simulator screen below.*

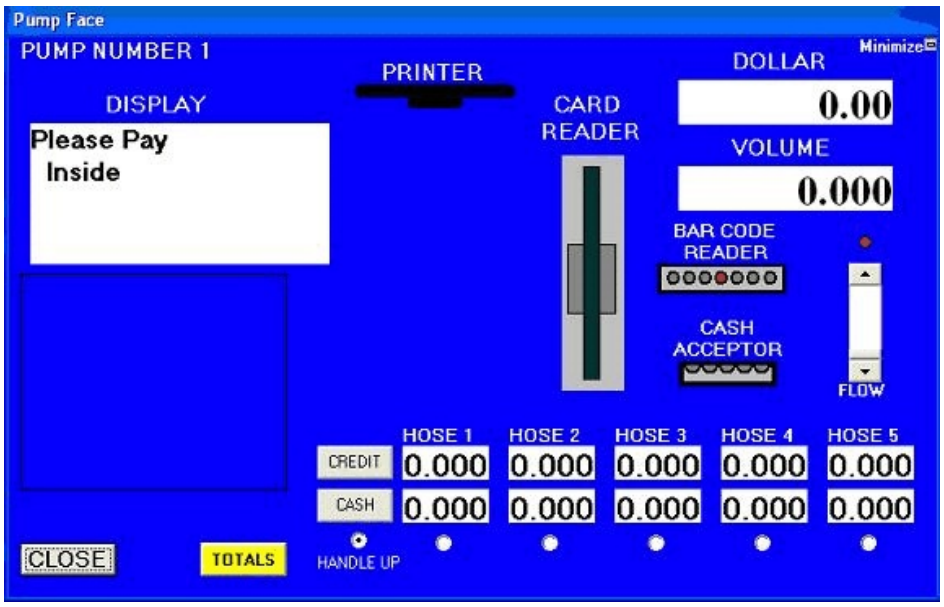


Click on number of the dispenser/reader to configure, then the PumpConfig button. *These choices will take you to the System Configuration window shown below.*



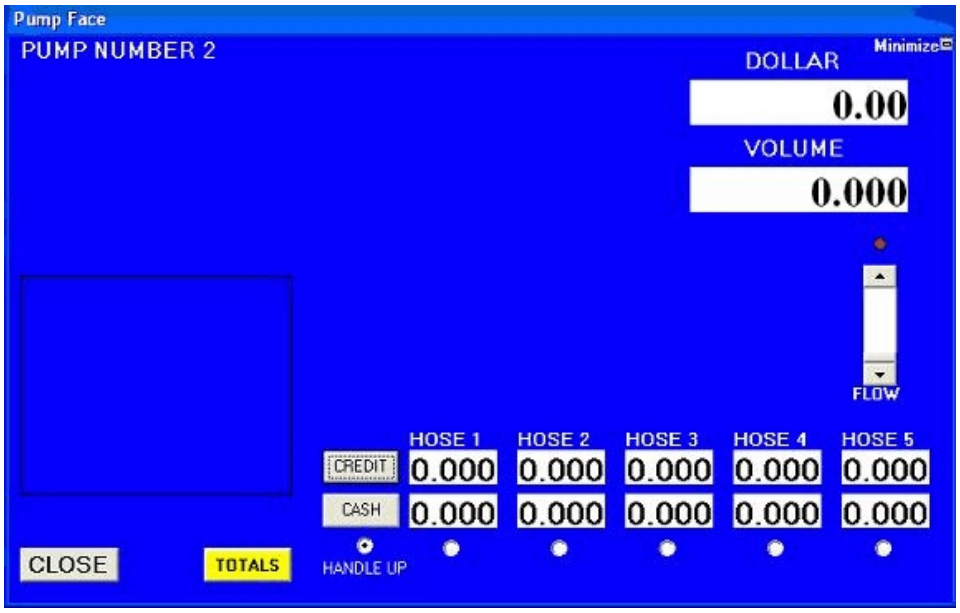
From the System Configuration menu, select the dispenser type and whether or not the reader is to be enabled. Then click on OK. After clicking OK, the Pump Simulator screen reappears. The button for the dispenser number already configured will have changed color. Continue until all dispensers to be configured have been completed and you have returned to the Pump Simulator screen.

At the Pump Simulator screen, click on one of the configured dispenser numbers and a Pump Face screen will appear. At this point, a simulated operation of dispenser and reader can begin. *See Pump Face screen below.*

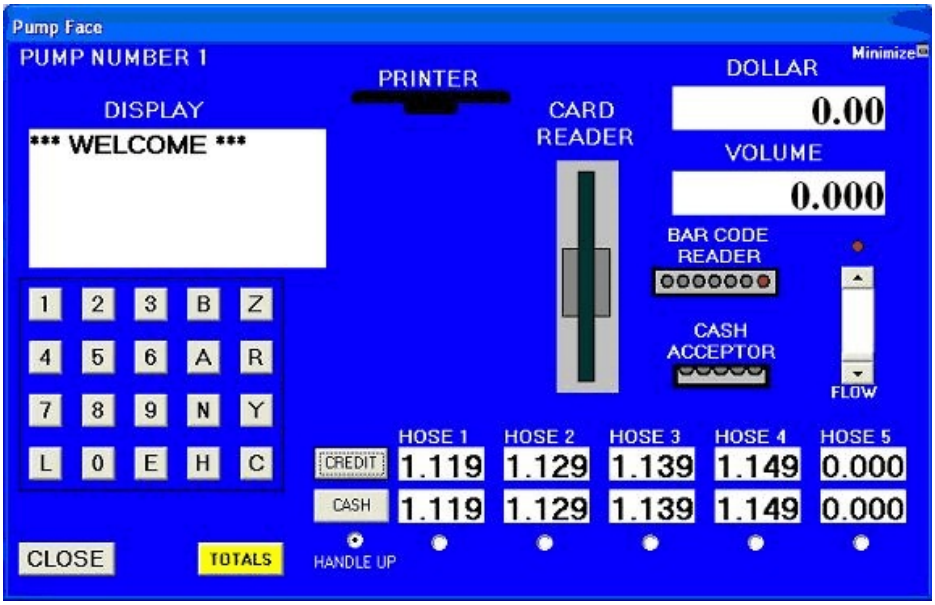


When your POS application has sent prices and reader key configuration to the PIE controller, the controller will in turn send that information to the pump simulator. The cash and credit prices will appear in the appropriate hose positions. The reader keypad layout (if present) will appear in the box under the display window. At this point, dispenser s and readers should be configured and the pump simulator program is ready to begin simulating transactions. If operation problems occur, review price and keypad configuration information. Incorrect data sent to the controller can produce unreliable operation.

See Pump Face screen below, demonstrated with no reader, as it will appear after prices have been sent.



See Pump Face screen below, demonstrated with reader, as it will appear after prices have been sent.



Operate Pump Simulator Program

The dispenser and reader simulator program is comprised of the following basic elements:

- Pump Number Window — Shows the active dispenser number
- Dollar Display Window — Shows the dollar amount of fuel dispensed during a transaction
- Volume Display Window — Shows the volume amount of fuel dispensed during a transaction
- Flow Control Bar — Variable rate flow control which allows the user to simulate the opening of the flow lever on the nozzle
- Hose Price Windows — Show the cash and credit prices per unit for each hose on the dispenser
- Cash/Credit Buttons — Allow the user to select the method of payment to be used for the transaction
- Handle Select Buttons — Allow the user to select hose to dispense product
- Reader Display — Shows all reader-related display messages
- Printer — Displays a simulated printer receipt
- Card Reader — Simulates entering a card, providing a selection of seven different card types.
- Keypad Buttons — Allow the user to enter keypad data
- Close Button — Allows the user to close the active dispenser/reader screen
- Totals Button — Displays all hose prices and totals

Upon exiting the pump simulator, the user will be prompted to save the current configuration. Current settings may be saved in a file to be reloaded at the next startup. *See Select a Configuration File screen below.*