

# CounterPoint EMV Processing

---

(CP-EMV)

Version 5.0x

**Ed Sills**

**11/17/2021**

# Contents

- CP-EMV ..... 8
  - What Is It ? ..... 8
  - Freeway Commerce Connect (FCC) ..... 9
  - How CP-EMV Works..... 10
  - Document Conventions ..... 11
  - Features ..... 12
    - Partial Payments ..... 12
    - Offline Authorizations..... 12
    - Tokenized Customer Cards ..... 13
      - Setup Customer Cards ..... 14
      - Tokenize existing customer cards ..... 16
      - Create Tokens from history ..... 16
      - Purging Customer Cards..... 17
    - Givex Gift Cards..... 17
    - CounterPoint File Utilities ..... 18
      - Compact Files ..... 18
      - Export in Two Directions ..... 18
    - Signature Capture ..... 19
    - Portable View EMV Transaction History Program ..... 22
    - Sounds..... 25
    - Securing Existing Credit Card Information ..... 27
      - Mask the information ..... 27
      - Purge the existing customer cards..... 27
    - FCC Test Client ..... 28
    - FCC Client User Interface (FCC-UI) ..... 31
  - Installation and Configuration ..... 33
    - Freeway Commerce Connect (FCC) ..... 33
      - USB Drivers for the Ingenico ..... 34
      - Freeway Commerce Connect Components ..... 34
        - Installing FCC Interactively..... 35
        - Installing FCC From the Windows Command Line ..... 40
        - Susanna Configuration File Upgrade Utilities ..... 43

Post-Installation Steps .....	49
FCC Server Service (Figaro) .....	50
Figaro's Configuration File .....	51
FCC Client Service (Susanna) .....	56
Susanna's configuration file .....	56
Servers.xml.....	64
Updating the Ingenico.....	65
FileWrite (versions 1.0 and 2.0) .....	65
POI Application Loader (PAL) .....	68
FCC Services Control Scripts.....	74
Figaro Service Control Scripts .....	74
Susanna Service Control Scripts .....	74
FreedomPay Service Manager .....	75
CounterPoint Installation and Setup.....	77
Creating a Verified Backup of your Existing CounterPoint Installation .....	77
Installing the Modified Programs and Files .....	78
Windows® Installation Instructions .....	78
Unix Installation Instructions .....	79
Back to Common Installation Instructions.....	80
CP-EMV.REG .....	80
Upgrading CP-EMV.....	81
Installation .....	81
Significant Changes From Version to Version .....	82
Version 3.x to Version 4.0 .....	82
Version 4.0 to Version 4.1.....	83
Version 4.1 to Version 4.2.....	83
Version 4.2 to Version 5.0.....	83
Updating CP-EMV.....	84
Upgrading FreedomPay .....	85
Upgrading Figaro.....	85
Upgrading Susanna .....	85
Modifications to CounterPoint .....	86
The New EMV Information File (SYEMVF) .....	86
Messages Displayed During Transactions .....	86

Setup>System>Company .....	87
Setup>Point of Sale>Control.....	87
Setup>Point of Sale>Stores>Stores .....	87
Setup>Point of Sale>Stores>Configuration options .....	89
Setup>Point of Sale>Forms.....	92
New EMV Tender Fields.....	92
Enhanced Form Field Process .....	94
Point of Sale Forms Printing Program.....	94
Setup>Customers>Control.....	95
Setup>Order Entry>Control .....	97
Point of Sale>Tickets>Enter .....	98
Point of Sale>End of Day>Post.....	99
Point of Sale>Offline operations>Create upload file .....	99
Point of Sale>Offline operations>Import upload file.....	99
Point of Sale>Touchscreen.....	99
Sales History>View>Ticket history .....	100
Customers Cards Maintenance Window .....	101
Customers>Customers.....	104
Customers>Cash Receipts>Enter .....	105
Customers>Cash Receipts>Edit list.....	113
Customers>Cash Receipts>Post.....	113
Customers>Month-end>Close open items.....	113
Customers>Change open items>Change open items.....	113
Customers>View>Open items .....	113
Customers>View>Closed items .....	113
Customers>Reports>Customer cards .....	114
Customers>Reports>Open item detail .....	118
Customers>Purge>Closed items .....	118
Customers>Purge>Customer cards .....	119
Customers>Utilities>Open item maintenance .....	121
Order Entry>Orders>Enter.....	122
Order Entry>Deposits and refunds>Enter .....	127
Order Entry>Deposits and refunds>Edit list .....	134
Order Entry>Invoices>Authorize.....	134

System>View>EMV Transaction history .....	135
System>Reports>EMV Transaction history.....	138
System>Reports>EMV Reconciliation report.....	140
System>Purge>EMV Transaction history.....	141
System>Utilities>Environment .....	142
System>Utilities>Ad hoc authorizations.....	143
DataLookup for EMV Transaction File .....	148
Add a Customer On-The-Fly (Point of Sale and Order Entry) .....	149
File Utilities (All packages) .....	149
Compacting Files .....	149
Automatic File Utilities.....	152
Exporting Files.....	152
Handling Locked Records While Exporting a File.....	158
File Utilities>Customers .....	159
File Utilities>Order Entry.....	159
File Utilities>Point of Sale .....	159
File Utilities>System.....	159
File Utilities>Special>Customers>Create tokens from history .....	159
File Utilities>Special>Customers>Set customer allow tokens .....	161
File Utilities>Special>Customers>Tokenize customer cards.....	162
File Utilities>Special>System>Create a new company .....	166
File Utilities>Special>System>Set CP-EMV fields.....	167
CounterPoint Startup Program .....	170
SCREEN Console I/O Program .....	171
POS Receipts .....	174
Offline Ticket Entry .....	174
Creating Workstation Files.....	174
Backup the Offline Data .....	175
Create the Upload File on the Offline System .....	176
Upload File List.....	176
Upload the Offline Data .....	176
Multi-Site .....	177
Let's Get Geeky !.....	179
The New EMV Information File.....	179

Changes to Existing CounterPoint Files.....	182
CounterPoint SQL Connection .....	182
Changes to Existing Tables .....	182
The New EMV Information File .....	183
Debugging .....	187
Restarting Figaro and Susanna.....	188
Log Files.....	190
Figaro Log File .....	190
Susanna Log File.....	193
Event Viewer Messages .....	194
Duplicate Charges .....	194
Contact Information.....	195
Sample Reports .....	196
Customers>Reports>Customer cards .....	196
Customers>Purge>Customer cards .....	196
By Parameter .....	196
By Position.....	197
Customers>Cash Receipts>Edit list.....	198
Customers>Reports>Open item detail .....	199
System>Reports>EMV Transaction history.....	200
Micro Format .....	200
Summary Format .....	202
Brief Format .....	203
Full Format .....	204
System>Reports>EMV Reconciliation report.....	206
Brief Format .....	206
Full Format .....	207
FileUtilities>Special>System>Mask credit cards.....	208
File Utilities>Special>System>Set CP-EMV fields.....	210
File Utilities>Special>Customers>Tokenize customer cards.....	211
File Utilities>Special>Customers>Set allow customer cards .....	212
File Utilities>Special>Customers>Create tokens from history .....	213



## CP-EMV

### What Is It ?

CP-EMV is a v7 CounterPoint modification by Infinity Software Solutions, Inc. (in conjunction with FreedomPay, Inc.) that replaces CounterPoint's built-in credit card processing with a system that satisfies the EMV liability shift and takes CounterPoint completely out of PCI compliance scope. It also allows merchants to accept other forms of payments such as prepaid cards, debit cards, processor-based gift cards (not CounterPoint gift certificates which still operate separately) and ApplePay®.

As of October 1<sup>st</sup>, 2015 major credit card processors shifted the liability for credit card fraud to the party who is the least EMV-compliant. So if a bank hasn't supplied EMV credit cards to their customers then they are the most liable for the fraudulent charges. But if the bank has issued a customer an EMV-capable card and a merchant swipes a clone of that card, the merchant will be liable for the charges. This shift in liability is the driving force behind the push to implement EMV credit cards nationwide.

EMV stands for Europay, MasterCard and Visa. These are the companies that created the original standard. For decades credit cards have used a magnetic stripe to carry the account information. However, the formatting of the data on these stripes is widely known and the hardware to recreate credit cards using stolen data is easy to obtain. Processors have been printing CVV values on the backs of their cards for years, but that isn't a solution for all forms of credit card fraud. EMV cards add a miniature circuit to the card to carry the account information. Looking at the front of the card you can see the contacts for this circuit on the left side of the card. This circuitry can't easily be replicated so EMV cards are inherently far more secure than magnetic stripe cards. Rather than swiping the card in the traditional manner, an EMV card is inserted into the terminal (called "dipping") and remains in place until the authorization process completes.

As previously mentioned, CP-EMV replaces CounterPoint's native credit card processing named CP-Gateway. Prior to this enhancement, CounterPoint collected the raw credit card information directly from the credit card terminal. That put CounterPoint in the position of having to meet PCI compliance regulations. So CounterPoint had to employ complicated encryption routines to encrypt the information prior to using the data or storing it in data files. If the encryption requirements were to ever change, CounterPoint would need to be recoded to meet the new requirements. This would be a problem since NCR has sunset v7. "Vanilla" CounterPoint encrypts the information only after it has been passed to it via the workstation's operating system. But that communication can easily be hijacked between the hardware and CounterPoint or from within the operating system. Thus even a fully up-to-date CounterPoint installation has a major vulnerability.

CP-EMV completely removes the encryption vulnerability and the merchant's liability for cloned magnetic stripe cards. With CP-EMV, when a customer tenders using a credit card CounterPoint merely asks the credit card terminal for a tender of the desired amount. The terminal's software accepts the tender from the customer, immediately encrypts it then obtains the authorization. It then provides to CounterPoint the pass/fail, authorization code, the masked credit card number and expiration date as well as a few other bits of information. Since CounterPoint never sees any sensitive credit card information it is completely "out of scope" of PCI compliance.



## **Freeway Commerce Connect (FCC)**

Freeway Commerce Connect is the product that provides the “backend” of the entire authorization process. It is owned, maintained and administrated by FreedomPay Inc. To use CP-EMV you must first contact a FreedomPay dealer (such as ourselves) and create a merchant account with them. Once you have signed up your FreedomPay dealer will provide you with your FreedomPay merchant credentials and the information you need to access the FreedomPay Portal for downloads and updates.

All of the FCC components are delivered in a single executable file. At this time there are four components to FCC.

- Freeway Server Service (Figaro)
- Freeway Client Service (Susanna)
- Freeway Client User Interface (FCC-UI)
- Freeway Test Client (FTC)

Notice the first two components have the names “Figaro” and “Susanna”. You might see these names in FreedomPay’s documentation, but they are phasing these references out. However these are the names we all used as the system was being created and we’re used to using them. Frankly it’s easier (and more clear) to refer to “Figaro” and “Susanna” than “Freeway Server Service” and “Freeway Client Service”. So we’ll be referring to Figaro and Susanna many times in this document. Get used to that now. The terms “server” and “client” are used so often and for so many purposes in our industry that we just find it easier and more concise to call them by their names – not their official titles.

**Freeway Server Service (Figaro)** is a Windows® service that dispatches all authorization requests. CounterPoint sends all requests to Figaro which repackages them and passes them on to the appropriate Susanna based on the workstation-id that’s included within the authorization request. The computer running Figaro doesn’t necessarily need to be dedicated to running only Figaro or even be a server-class machine. Of course it should be dependable and be secured with antivirus and a firewall.

**Freeway Client Service (Susanna)** is a Windows® service that typically runs on the register with the Ingenico attached to it via a USB cable. It communicates with Figaro and controls its attached Ingenico. It performs the authorization except for voids, tokenized transactions and offline authorizations (which are discussed later). I say “typically” because there is another seldom used scenario. We’ll discuss that later in “How CP-EMV Works”.

**Freeway Client User Interface (FCC-UI)** is a program that runs on the Susanna machine that presents a status window to the clerk as the authorization process takes place. It also has the ability to perform several connectivity tests and can initiate updates to the Ingenico. Generally it just keeps the clerk informed as to what is happening on the FreedomPay side of the processing.

**Freeway Test Client (FTC)** is an optional FCC component that allows running authorization requests outside of CounterPoint. It’s a good test tool for determining whether a problem exists within or outside of CounterPoint. However since it has the ability to perform actual authorizations you probably don’t want to install it at each register. Typically you could configure it to authorize against FreedomPay’s UAT server (a test transaction system) in which case it wouldn’t be so bad; as long as someone doesn’t change the settings or use the program thinking they’re authorizing credit cards for real.

## **How CP-EMV Works**

How CP-EMV is installed and implemented depends on the platform CounterPoint is running on. But there are some commonalities between the platforms. Each enterprise (or store) runs a FreedomPay program named “Freeway Commerce Server” – a.k.a. “Figaro”. Figaro runs as a service on a Microsoft Windows® computer. Figaro is the Windows® service that CounterPoint sends the authorization requests to. Typically it runs on a networked server at the store (one per store). If your system is running across a WAN or on a COLO environment you may have a single Figaro server handling all of your stores. If you intend to use a single Figaro to service several stores you could install it on “the cloud” to improve redundancy. Or Figaro and Susanna could both be installed on the register if you only employ a single register.

Typically, each Windows® based POS register has an Ingenico credit card terminal plugged into it via a USB cable. Each of these machines runs a FreedomPay program named Susanna which controls the Ingenico on that machine. However, there is another seldom used configuration. Some larger enterprises might not use Windows® based computers as registers. They use networked “dumb terminals” which are basically just consoles that have a display and a keyboard. Since they aren’t computers they cannot run Susanna. In this environment the store would use Ethernet-connected Ingenicos (as opposed to USB-connected). All of these Ingenicos and the dumb terminals are connected to the local network. Each store has a single Windows® based computer running Susanna controlling all of the Ingenicos. Refer to the diagrams at the end of this document for depictions of typical installations.

When a register is turned on its Susanna service starts, connects to Figaro (via the network) and sends Figaro a message identifying what workstation-id it is. (The workstation-id is defined in Susanna’s configuration file.) From the incoming network communication Figaro knows what network IP address that Susanna is at. Figaro builds an internal table of what workstation-ids are attached to it and their associated IP addresses. When a register requests an authorization, CounterPoint communicates to Figaro (via the network) passing to it (among other things) the register’s workstation-id and the tender’s requested amount.

Using the workstation-id Figaro looks into its internal table to find the IP address of the register. It sends the request to the register’s Susanna to wake up the attached Ingenico and accept a tender for the desired amount. The Ingenico prompts the customer for the credit card information. The customer provides it by swiping the magnetic stripe, inserting an EMV card, manually keying the card information or waving a near-field communication device such as an RF card, ApplePay or a Google Wallet device. The Ingenico immediately encrypts the information. This prevents a thief from installing a wedge device in the cable between the Ingenico and the register to scrape credit card information (which was completely possible in CP-Gateway). The Ingenico sends the encrypted information to its Susanna which performs the authorization via encrypted communication directly to FreedomPay’s gateway. If the card was declined, the customer cancelled the transaction or there was some other error Susanna notifies Figaro which sends the result to CounterPoint to handle as required. If the authorization is accepted and the Ingenico is signature-capable, Susanna requests the Ingenico to prompt for the customer’s signature (if desired). The results of the authorization are then sent to Figaro to pass back to CounterPoint. Since neither the auth request from CounterPoint or Figaro nor the response back from Susanna contains any account specific data these communications are not encrypted.

## Document Conventions

This document has many references to menu selections, screens, fields and commands. To discern them from normal text and provide a standardized nomenclature we've utilized some conventions.

Menu selections are depicted in *italics*. All menus in CounterPoint are multi-level meaning that you need to select at least two menu entries to access any program. Our convention to depict these selections is to string them together with a "greater than" symbol between them. So to depict the menu selection for the Customers control record maintenance program we would use *Setup>Customers>Control*.

If the program has multiple screens we append the menu selection with either the title of the screen or the number of the screen if it has no title like this *Setup>Customers>Control>"Customer cards"* and this *Setup>Customers>Control>screen#-1*.

To depict a specific field on a screen we append the menu selection with the field name in double quotes like this *Inventory>Items>"1.Item number"*. If the program has more than one screen we use the nomenclature above then add the field's name like this *Setup>Customers>Control>"Customer cards">"1. Use customer cards ?"* and *Setup>Customers>Control>screen#-1>"2. Next customer number"*. Notice we don't use different notations for titled screens and fields. If the text has two segments enclosed in double quotes it should be obvious it's depicting a screen and a fieldname. If there's only a single segment enclosed in quotes, we're referring to the entire screen or the program only has a single screen and the segment refers to a fieldname or the program has multiple untitled screens, but all of the fields are uniquely numbered. We're certain you'll become accustomed to the standard.

Commands entered on a command line are depicted in Courier font. Sometimes we use a small font size to get the command to fit on a single line. They are often depicted on their own line to separate them from the normal text and to make them look like they would in a command line session like this.

```
Install_EMV c:\syn
```

This document makes extensive use of hyperlinks to allow the reader to jump to another section of the document without having to find it or use the index. A hyperlink could be any word or phrase. They are indicated by being colored [blue](#) and being underlined. (They may change from blue to purple once you've clicked on them.)

At the bottom of nearly every screen in CounterPoint you'll find the ubiquitous "Field number to change ?" prompt. We refer to this prompt a lot in this document and frankly it's really annoying to type over and over again. So we're not doing it. Whenever you see "FNTC" understand we're talking about the "Field number to change ?" prompt.

Keystrokes are denoted between less-than and greater-than symbol like this "<ENTER>", "<ESC>", <F1> and "<CTL+F1>".

Generally, whenever we use a pronoun for a person we use "he", "him" or "his". That's not because we think the world is composed of all men. Nor do we think that men run everything. It's just really tiresome to put "he/she", "him/her" or "his/hers" throughout the document. And being a dinosaur I am not up on the new fangled non-binary pronouns. (Did I even say that right ?)

## **Features**

### **Partial Payments**

In 2010 MasterCard and Discover started requiring merchants to support partial authorizations for debit cards, prepaid cards and gift cards. Stiff per-incident fines have been levied for failing to support partial payments. The partial payments requirement is actually threefold.

- The software must recognize and accept a partial payment from the processor.
- The software must allow the customer to substitute a different card in lieu of one already authorized. The prior card's authorization must then be voided.
- The software must have the ability to print the card's remaining balance on the receipt.

This enhancement adds the ability to handle all three requirements. When an authorization is requested and the result is a partial payment the program will display a different message resembling "Partial authorization accepted for \$12.34 – *auth-code*". If you have assigned a wave file (discussed below) to partial payments CounterPoint will play the sound. The amount tendered and total remaining is automatically adjusted to reflect the amount authorized. At this point the clerk should prompt the customer for another tender, void the ticket (which voids the authorization) or handle the ticket in some other manner.

### **Offline Authorizations**

FreedomPay's FCC has the built-in ability to perform offline authorizations. If Susanna cannot connect to the gateway it passes back a particular error code to Figaro. If you are allowing offline authorizations and the amount to be authorized is less than the floor limit, Figaro will store the authorization request in its own database, create a "fake" auth code and request-id and pass these values back to CounterPoint as if the authorization had been accepted. Then Figaro will periodically attempt to get a "real" authorization for each offline authorization. Of course by then CounterPoint has moved on. The pass/fail status of the "real" authorization can be viewed on the merchant's FreedomPay portal website. If you're willing to accept the possible liability of a few charge backs you might want to take advantage of this feature.

See the section "[Installation and Configuration](#)" for details on setting up this feature. There are only a few settings in Figaro's configuration file to be made.

## Tokenized Customer Cards

Storing a customer's credit card information in your POS system can be very useful. For some businesses such as B2B or A/R only businesses it is essential. But having customer credit card information stored in your POS system is a huge liability. So how can you tender a ticket (or a cash receipt or a payment on an order) without the card being present if you can't store your customer's card information ?

Originally POS systems would merely store the card information in the database's native format. After all, most systems store their data in a compressed format that is indecipherable to all but the most cunning programmer. When the credit card industry shifted liability and began implementing fines to merchants whose POS system stored credit card data, the obvious solution was to encrypt the credit card information. That's what Synchronics did with CounterPoint.

Synchronics did what they needed to pass an expensive PCI audit to allow retaining encrypted customer credit card information in CounterPoint. Not only did they encrypt the credit card information stored in the customer record they also encrypted the credit card information in every file that carried credit card from the moment a card is tendered through posting and into the sales history file. But they also managed to slip through a huge, glaring hole in their compliance. They didn't encrypt the credit card information stored in the POS Tracking file. You can key a single credit card on the customer maintenance screen. That gets encrypted, but if you press <F7> and enter (up to) 8 more cards, that card data is not encrypted at all. Anyone can view the numbers regardless of whether they are authorized to view unmasked credit card numbers or not. If you export the POS Tracking file you'll get a file potentially loaded with customer credit card numbers and expiration dates. I was really surprised when I saw this. So if you have any additional customer credit cards on file beside the single credit card entry on the main customer maintenance screen, you are out of PCI compliance and living on the edge.

So what should you do with your existing liability ? CP-EMV has many options on how to handle the credit card information you have scattered throughout CounterPoint. You can leave the historical information as-is or mask it, tokenize the existing cards or purge them. Each option has its own full-blown description later in this document.

Eventually the credit card processing industry came up with a better solution than encrypting. Encryption is good, but any encryption scheme can eventually be broken or its key leaked. So the concept of tokens was created. A token is a seemingly nonsensical string of characters that represents the credit card it is associated with. It's not an encryption of the original information so it cannot be unencrypted. It's more of a place-marker that only works within the processor's own database. By replacing encrypted credit card information with tokens within his POS system a merchant shifts the liability for their customers' card data to the processor.

FreedomPay allows creating a token for each credit card it processes. In fact, it allows creating a token for a customer's credit card without obtaining an authorization. That token can then be used to tender a future authorization request. Tokens are only good for two years. That expiration date is based on the current date and has nothing to do with the underlying credit card's expiration date. So it's not uncommon for a token to have an expiration date that is later than the credit card it represents. The assumption is that the card number will be renewed at some point and so the token will still be valid.

We have set up CP-EMV to return a token with every FreedomPay authorization. It is one of the many fields that are returned with any successful authorization. We store all of this information in the EMV Transaction History file every time. The exception to this is when you obtain an authorization via a token (as opposed to tendering with a physical card). Basically, you cannot extend the life span of a token by using the token for subsequent authorizations. Storing all this EMV information for each authorization can be quite useful. You can use it to continuously update your customers' tokenized cards on file. You can also use it to generate tokens for your customers' cards from EMV history.

### *Setup Customer Cards*

CounterPoint has allowed storing customer credit card information for a long time. But their solution was a completely manual process. Someone would need to enter the credit card information. Then if the customer wanted to store a different (or additional) card, that card would need to be keyed. If the customer's card expired someone would need to rekey the updated card information. You can still do that with CP-EMV if you like it. But an alternative to manually maintaining your customer's customer card information is to allow CP-EMV do it for you. It's such a nice feature I can't believe CounterPoint didn't already have it. Every successful EMV transaction returns a token with a fresh token expiration date. So it's programmatically possible to update a customer's card information with every successful transaction.

CP-EMV doesn't automatically save customer cards without your permission. You need to turn it. You also need to enable customer cards and set them up. You can find the information on that in your CounterPoint documentation. You need to decide if you want to automatically update card information for all of your customers or on a case-by-case basis. Some merchants need to have their customer sign a consent form before they will save card information for the customer. That might be prudent, but it might also be a bit of overkill. After all, you are not storing actual card information. You're storing a token that represents the card on FreedomPay's credit card database server.

Let's touch on how to configure customer cards in order to save customer tokens.

### *Define Customer Cards*

As already mentioned (and rather obvious) you must have customer cards enabled. This is done in [Setup>Customers>Control](#). I'm not going to discuss how to set up customer cards here. You can find that information in the CounterPoint documentation. But will discuss what you need to know as far as CP-EMV is concerned.

If customer cards is not enabled you're already ahead in the game. The customer card "slate" is clean and you don't have to worry about migrating or purging existing card information. Turn on customer cards and setup the number of EMV pay codes you want to use.

But if you have been using customer cards for any length of time you're going to need to think and maybe even plan ahead. Hopefully you haven't already set up all 8 entries leaving none for EMV pay codes. You probably want to convert your existing customer cards to EMV tokens. The old CP-Gateway style credit card entries will not work with CP-EMV. The whole point of CP-EMV is to eliminate the stored card liability (and to be able to process credits cards within CounterPoint). Ultimately you'll probably want to tokenize your existing customer cards then purge those old customer cards.

If you're going to throw away all of your existing customer cards don't do it here by merely changing your existing customer card definitions. Changing an entry from a CP-Gateway pay code to an EMV pay code doesn't do anything with the customer card data you already have on file. You'll basically create trash data. You need to purge the customer card data associated with those old customer card entries before you can reassign the entry to be an EMV entry. We provide a nifty program to handle that the details of which you can find [HERE](#). Once all the old customer card data has been purged you can assign up to 8 of the entries on this screen to carry up to 8 credit/debit cards as EMV-enabled cards. Assigning all 8 slots as EMV cards probably isn't necessary. But you're a better judge of the number of cards your customers might want to keep on file.

So let's say you want to keep your existing customer card data at least until you've created tokens from them. Ideally you'll have at least as many empty slots as you have CP-Gateway style customer cards defined. Otherwise you're going to have a challenge ahead. Assign an EMV pay code to at least as many slots as you have CP-Gateway style customer cards defined. At this point you have your old CP-Gateway customer cards AND your new EMV customer cards on the same screen. This is when you run our program that tokenizes your existing customer cards. The details on that are [HERE](#). Having run that program (and verifying it worked OK) you can then purge the old customer cards (details [HERE](#)). BTW – the customer card purge program has the ability to delete the old customer card entries it tokenizes as it runs. That could save you a step once you're comfortable with the process.

Once you're done the customer card definitions might look something like this.

```

Control                                                                    7.5.20
                                     --- Customer Cards ---
1. Use customer cards ?                Y
2. Customer card types
   Card-name  Exp  Pay-code
   Discover   Y    7    Discover
   Mastercard Y    3    MasterCard
   Uisa       Y    6    Uisa
   EMU1       Y   15    EMU Paymnt
   EMU2       Y   15    EMU Paymnt
   EMU3       Y   15    EMU Paymnt
3. Default entry mode                  Magstripe
4. Auto-save customer cards            Always
5. Allow payment with card
   even if card not swiped ?          Y
Field number to change ? 

```

Having defined the EMV customer card entries you need to determine if and how you want to automatically save future customer cards for your customers. That is accomplished on the same screen via the new field "4. Auto-save customer cards". This field controls whether a token that is returned with successful authorizations is saved with the customer's cards. It allows three values.

- "Always" Always save any token received as a customer card.
- "By customer" Save the token as a customer card if the new "Tokens?" field on the customer maintenance screen has been set to "Y".
- "Never" Don't save tokens for any customers.



When CP-EMV updates a customer's cards it uses the masked credit card to check if the card is already present in an EMV customer card slot. If it is, CP-EMV updates that entry. If it's not present and there's an empty EMV customer card slot it adds the tokenized card there. There is one customer whose customer cards will never be updated. That's the default customer# that's defined for each store (i.e. the "Walk-in customer").

### **Enabling Tokens for Individual Customers**

If you indicated "By customer" in the Customers control record (above) you'll need to populate the new "Tokens ?" field on the customer maintenance screen before that customer will start automatically saving tokens. You can see details on that new field [HERE](#). If you indicated "Always" or "Never" the contents of this field is immaterial. The "Tokens ?" field allows the values "Y" and "N", but the only value that really matters is "Y". Any other value is interpreted as "N".

If you have a thousand customers on file and you need to turn this flag on for 90 percent of them manually setting this field customer-by-customer would be highly annoying. To help you set this new field en mass we created a program that will turn the flag on or off depending on user-defined parameters. You can see the details of the program [HERE](#).

### **Tokenize existing customer cards**

As previously mentioned, installing CP-EMV doesn't mean throwing away the customer card data you've collected over the years. You can convert this credit card information to tokens and use them going forward. Tokenizing your existing customer cards will create a new token customer card entry for each of the customer credit cards you have on file. You can see the details of the program [HERE](#).

The program first tokenizes the credit card that stored on the customer record. Then it tokenizes the additional cards in the POS Tracking file. The program will optionally remove the old credit card from the files as they are successfully tokenized. So at the end of the process you could have no unencrypted credit cards in your system at all. They would have all been converted to tokens.

Keep in mind that tokenizing each card is similar to obtaining an authorization. Each requires sending a request to FreedomPay's gateway. So this program doesn't run nearly as quick as a typical report.

### **Create Tokens from history**

If you've been using CP-EMV for a while you probably have a good amount of EMV transaction history on hand. If you didn't initially set CP-EMV up to automatically save tokens for your customers you can still use this program to create tokens for your customers from that saved history. You can see the details on that process [HERE](#).

And BTW – Since this program uses information already obtained from FreedomPay – including token values – this program doesn't require a trip to the FreedomPay gateway. So it runs much faster than creating tokens from existing customer cards.



## *Purging Customer Cards*

At some point you are going to want to purge customer cards. They build up over time and they expire. The CP-Gateway customer cards are dead weight in CP-EMV since you can't use them as a tender. Their best use is to create tokenized customer cards from them then delete them. Keep in mind that those extra CP-Gateway customer cards are not encrypted. You definitely want to handle those ASAP.

We provide a utility program that allows purging customer cards based on a lot of parameters. In fact we provide two completely different parameter screens to purge card based on two completely different methodologies. You can find the details on this program [HERE](#).

If you're purging customer cards based on their expiration date(s), you'll want to use the first parameter screen. If you want to purge cards based on the position they have in the customer cards table - as in purging all your existing CP-Gateway cards - you'll want to use the second parameter screen.

In all of this customer card manipulation you need to keep in mind that you need to handle your customer card data BEFORE you make changes to the customer card definitions. If you change the customer card definitions while you still have cards on file for those slots in the table, you could make the existing cards inaccessible or complete trash and make it impossible to remove them after you've started adding new cards to the redefined customer card definition slot.

## *Givex Gift Cards*

CounterPoint offers support for their own flavor of built-in gift cards. It's a good option and it works well for everyone - with a single store. The problem is when you start introducing multiple installations of CounterPoint. Even if they're running on machines one foot apart. Even if they're running on virtual machines on the same physical machine. The problem is the installations have different sets of data files and they don't see what the other store is doing with gift cards (or any other data). At night the installations can run Multisite and get things back in sync - until the next day. If a customer buys a gift card at store# 1, store# 2 won't know about the card until MultiSite has run. So a customer can't buy the card at store# 1 then drive over to buy something at store# 2. Even worse, a customer can redeem most of a gift card at store# 1 then drive over to store# 2 and redeem the full amount there. That's a problem.

There are other gift card solutions out there, but they're all external to CounterPoint so a merchant has to do some weird things in CounterPoint to support them. But since these solutions are web-based the problem of keeping gift cards in sync between stores is eliminated. In CounterPoint these types of cards are called "Stored value cards" or "SVC" cards.

We solved all this by integrating Givex gift cards into CounterPoint as part of CP-EMV. Givex is a web-based gift card solution that FreedomPay supports. So the cards basically act like a credit card except you can buy them, recharge them and redeem them.

The first step to utilizing Givex cards in CounterPoint is to open an account with them. Once that's done they'll provide you with the details you need to pass on to FreedomPay when you board with them. CP-EMV doesn't have any additional setting to support Givex cards.

Otherwise Givex cards work like any other SVC in CounterPoint. We're not going to rehash how to define and use SVC cards within CounterPoint. You can find all of that in CounterPoint's documentation.

## CounterPoint File Utilities

Even though CounterPoint's file utilities are clearly not part of authorizing credit cards we did bundle some big changes to the file utilities in with CP-EMV. These are changes that are large enough to be labeled a feature. For more information on these features please refer to "[File Utilities \(All Packages\)](#)". But we'll outline them below.

### *Compact Files*

We added the ability to compact any file in the CounterPoint file utilities. When a file gets too large or too slow the official solution is to export then import the file. This reduces the file size (by excluding deleted records) and rebuilds the file's indices. The problem is sometimes the resulting export file is too large for the operating system to handle. Or the time it takes to export then import is more than what's available.

CP-EMV solves this by writing the file to a new file with the exact same format then renaming the files so the new file becomes the production file. This requires only a single pass through the original file – no importing step – and you're done. This is a huge benefit and frankly, a lot of tedious code.

### *Export in Two Directions*

Another addition to the file utilities is the ability to export files forward or backward. Normally the file utilities export a file by reading the next record then the next record in a forwards direction starting at the beginning of the file (or whatever value the operator indicated as the starting key). Reading the file from the end of the file towards the beginning allows one to get passed a corrupt section of a file losing as little data as possible.

## Signature Capture

It is up to the merchant whether they want to collect signatures from their customers. Depending on their industry they might be required to collect them. They might want to get the customer's signature for tenders that exceed a preset threshold. FreedomPay built this capability into their software out-of-the-box. Of course, to collect signatures you'd need to have Ingenicos that allow for signature capture.

FreedomPay's software collects all of the signatures from the Ingenicos and makes them available on their Enterprise Portal. (For answers about the Enterprise Portal contact your FreedomPay dealer.) But that doesn't really provide the immediate gratification some merchants like. It also makes it a bit inconvenient to show a customer. It's a lot easier the spin the monitor towards the customer than to find a manager with the credentials to open a browser, log in to the Enterprise Portal, find the transaction and pull up the signature. CP-EMV allows the merchant to save the signatures locally and makes them quickly available in several areas of CounterPoint.

- [Customers>Cash receipts>Enter](#)
- [Customers>View>Open items](#)
- [Customers>View>Closed items](#)
- [Order Entry>Orders>Enter](#)
- [Order Entry>Deposits and refunds>Enter](#)
- [Sales History>View>Ticket history](#)
- [System>View>EMV Transaction history](#)

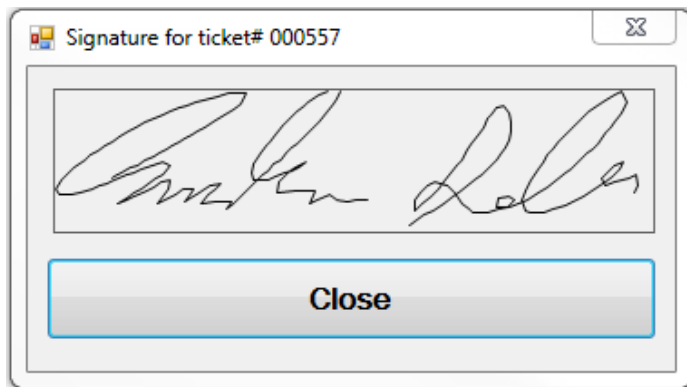
For more information on how we modified these programs please refer further down in this document where we discuss the changes to individual CounterPoint programs.

To enable capturing signature files locally go to [Setup>Point of Sale>Stores>Configuration options>"Credit Card Processing Options"](#) and set the field "6. Store signatures locally?" to "Y". Once this is set CounterPoint will begin saving the signature captures in the SYDATA\IMAGES directory. This is a new directory that didn't exist in CounterPoint before. When you change the field to "Y" the program checks to see if the directory already exists. If it doesn't the program creates it.

The format of the signature file is determined by the signatureFormat parameter in Susanna's configuration file. CP-EMV currently supports the parameter values "png" (recommended) and "scd1". The value "png" will create an industry standard image file that is viewable by most any image viewing program. The value "scd1" will create a proprietary file format which is easily understood, but doesn't have a viewing program. So basically, if you want to store signature files locally set the signatureFormat parameter to "png".

Starting with version 4.0x CP-EMV ships with a program named ISSviewer.exe that displays .PNG files. As you can probably tell by its name the program is a Windows® program. So it won't work in a Unix/Linux (\*nix) environment. (Frankly I haven't had time to create a matching program for that platform.) But that doesn't mean that you have to use our program to view the signature or that you cannot view the signature in \*nix. When a CounterPoint program attempts to display a signature it looks for the environment variable SigCapCmd. If it has been defined the program will use the value of the variable as the command to display the signature. So if you already have a favorite image viewing program you can create an entry in SYNRUN.SYN (Windows®) or synsupl (\*nix) to pass that program name into CounterPoint. If SigCapCmd isn't defined (or is blank) and you're running under Windows® the CounterPoint program defaults to the value "ISSviewer.exe".

A signature displayed in Windows resembles:



When a CounterPoint program makes the call to display a signature it builds a command line using the program to display the file followed by two parameters; the full name of the file to be displayed and (optionally) a string to be used in the title bar of ISSviewer.exe. Since having space characters embedded in such a string would cause the string to be passed as multiple individual parameters to ISSviewer.exe, we require that any spaces you want the string to include are converted to underscore characters. ISSviewer.exe replaces these underscore characters to spaces as the string is loaded to the title bar.

So a sample command line to display a signature would resemble:

```
ISSviewer.exe c:\syn\DEMO1\SYDATA\IMAGES\F9TYQ...YLGAAUG5.PNG Sigcap_for_ticket#_12345
```

Notice the name of the .PNG file ? Every credit card transaction creates a matching EMV information record carrying a unique 32-character request-id. We use this request-id as the name of the signature file. When a transaction is posted the associated EMV information record and signature file "follows" the transaction into history. When an EMV information record is purged (via [System>Purge>EMV Transaction history](#)) the associated signature file is also deleted.

If you want to quickly identify what value has been assigned to the SigCapCmd environment variable go to [System>Utilities>Environment](#) and press <F1> for "Advanced settings". You'll find it towards the bottom of the list.

BTW – While testing this feature I found myself repeatedly pressing the ESC or TAB keys to close the signature window. Of course being a Windows® application it really wanted me to click the “OK” button or press the ENTER key to accept the default form action (which is to click the “OK” button). So I tweaked ISSviewer.exe to allow pressing the ESC or TAB keys to also close the signature window. It works more naturally that way.

Note that the program that utilizes this program determines the function key that invokes it. These programs all have different function key assignments so we couldn't standardize on a single function key that calls it in every program.

So going forward in this document we are not going to spell out how to use this feature or what you'll see when you use it. We'll just mention it is available and maybe add a link back to this section.

## Portable View EMV Transaction History Program

CP-EMV comes with a full-blown, full-screen EMV transaction history program that allows viewing your EMV history filtered by many parameters. You can see the details on that [HERE](#). But that's overkill when you merely want to view the EMV history associated with a particular transaction in CounterPoint. Let's say you're viewing a customer's open items and you want to see the EMV transaction that authorized a particular payment. This would be impossible with vanilla CounterPoint. The ability simply didn't exist. We decided that was a nice feature that could be added. So we did.

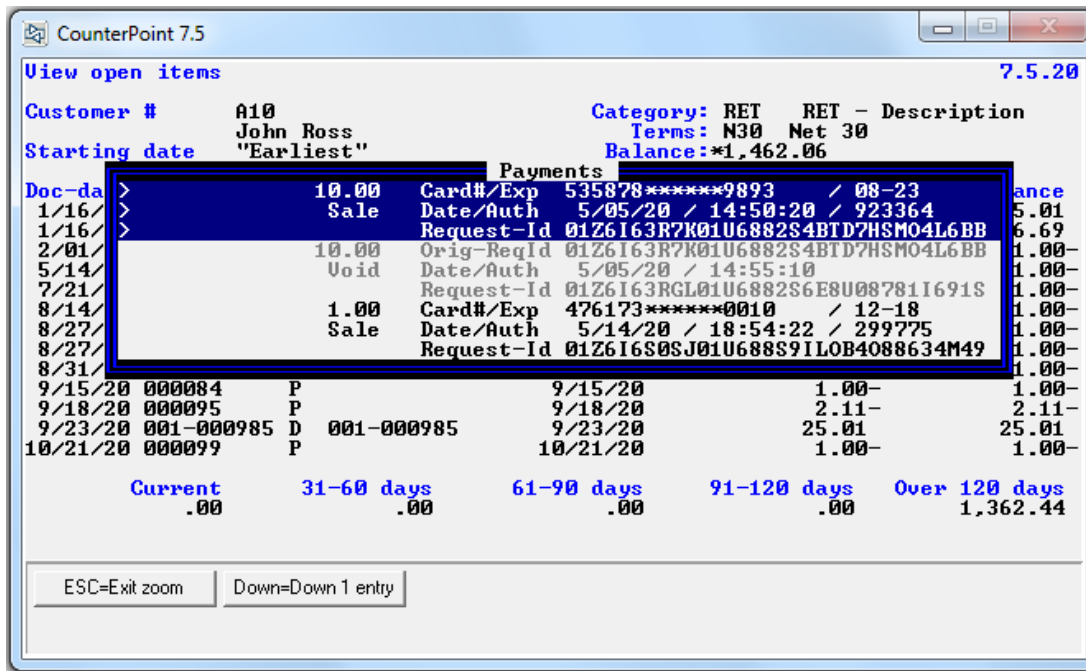
We wrote a separate program that allows viewing just the transactions that are associated with the transaction passed to it. This program can be called from any program in CounterPoint that we code for it. That's why we call it "portable". Because it is called from another program that already controls the screen we made this program open a window on the screen and display its contents in that. That way we don't affect what's already on the screen by the host program.

The format of the EMV information displayed is the same for all types of transactions, but some areas of CounterPoint (like ticket entry) might display more than just EMV history in the window. Here's an example of an A/R open item that has EMV history on file.

View open items							7.5.20
Customer #	A10		Category: RET				RET - Description
	John Ross		Terms: N30				Net 30
Starting date	"Earliest"		Balance: *1,518.06				
Doc-date	Doc-#	Typ	Apply-to	Due-date	Doc-total	Doc-balance	
1/16/19	001-000597	I	001-000597	2/15/19	25.01	25.01	
1/16/19	001-000599	I	001-000599	2/15/19	76.69	76.69	
2/01/19	000034	P		2/01/19	11.00-	11.00-	
5/14/20	000059	P		5/14/20	1.00-	1.00-	
7/21/20	000070	P		7/21/20	1.00-	1.00-	
8/14/20		P		8/14/20	1.00-	1.00-	
8/27/20		P		8/27/20	1.00-	1.00-	
8/27/20	000073	P		8/27/20	1.00-	1.00-	
8/31/20		P		8/31/20	1.00-	1.00-	
9/15/20	000084	P		9/15/20	1.00-	1.00-	
9/18/20	000095	P		9/18/20	2.11-	2.11-	
9/23/20	001-000985	D	001-000985	9/23/20	25.01	25.01	
10/21/20	000099	P		10/21/20	1.00-	1.00-	
Current		31-60 days	61-90 days	91-120 days	Over 120 days		
.00		.00	.00	.00	1,362.44		
<div style="display: flex; justify-content: space-between; border-top: 1px solid black; padding-top: 5px;"> <span>F5=Customer</span> <span>F6=Zoom</span> <span>F7=Age customer</span> <span>F8=EMV info</span> <span>ENTER=More info</span> <span>ESC=Exit</span> </div>							

Notice the <F8> prompt for "EMV info". If the program finds any EMV records for the entry it offers this key.

Pressing <F8> calls the portable EMV transaction viewing program for this transaction and you get:



If you're old enough you might remember "greenbar" computer paper. This window borrows on that concept. Each EMV entry is three lines on the screen. But as mentioned before ticket entry also display tenders other than EMV. Those entries can be as short as one line. Merely listing each entry down the screen makes it difficult to discern entries. Putting a blank line between them helped, but made the windows contents looking "flat" and uninteresting. Instead we color code entries in alternating grey and black characters. The currently selected entry displays with white characters on a blue background. For those unfortunates using a black & white terminal we provide greater-than symbols on the left edge of the selected entry. One can use the up/down arrows to navigate the list of transactions which can be up to 100 entries long. The arrow keys don't navigate up or down one line. They navigate an entire entry at a time. So navigating through the entries can be pretty quick.

The window sizes itself to fit the number of payments being displayed up to a point at which it then allows scrolling through the entries via the up/down keys. If the program finds a signature on file for the entry it offers <F1> to display it. You can find more on displaying signatures [HERE](#).

We added this feature to the following areas in CounterPoint.

- [Customers>Cash receipts>Enter](#)
- [Customers>View>Open items](#)
- [Customers>View>Closed items](#)
- [Order Entry>Orders>Enter](#)
- [Order Entry>Deposits and refunds>Enter](#)
- [Sales History>View>Ticket history](#)
- [System>View>EMV Transaction history](#)

Note that the program that utilizes this program determines the function key that invokes it. These programs all have different function key assignments so we couldn't standardize on a single function key that calls it in every program.

So going forward in this document we are not going to spell out how to use this feature or what you'll see when you use it. We'll just mention it is available and maybe add a link back to this section.



## Sounds

In a busy environment a clerk might not notice that the credit card they just processed didn't actually get an authorization. I've heard of clerks (using CP-Gateway) that would routinely receive an error message when processing a credit card for an order, ignore the message and continue right on as if it had worked. (Don't ask me how.) This is particularly prevalent in installations where CounterPoint is set up to display a message when an authorization is successful. Clerks just get used to receiving a message – good or bad - and ignoring them.

CounterPoint already has the option not display a message when an authorization is successful. But that still left other situations (declines, failures and partial authorizations) which each needed to have unique message. We pondered on how we could better inform the operator; basically make them look at the screen to see the message.

In a prior, unrelated modification we added the ability to play wave files when certain situations arose in CounterPoint. We thought that might be a great solution to the problem. After all, one cannot turn off their ears right? So we added the ability to assign a wave file to the 3 possible outcomes of attempting an authorization – accepted, declined (or an error) and a partial payment.

To this end we added optional, user-defined sounds to the authorization process for each of the three situations. They are independently controlled by three environment variables.

- **BadAuthWav** Carries the name of the \*.WAV file to be played when a card is declined or a failure is detected
- **GoodAuthWav** Carries the name of the \*.WAV file to be played when a card is successfully authorized
- **PartAuthWav** Carries the name of the \*.WAV file to be played when a partial authorization is accepted

CounterPoint issues the same sound for all messages. Merely adding additional sounds caused a conflict between the normal message sound and the authorization sound. So we knew we had to modify the routine that plays the sound whenever a message is displayed. It then occurred to us that we could further modify the routine to require the operator to press a specific key other than the usual ENTER key to acknowledge the message and clear it from the screen.

Putting the two together creates a pretty good solution. Good, partial or bad authorizations can all be configured to play a different sound and a partial or bad authorization requires pressing the <ESC> key to clear the message from the screen. If the authorization is successful you might want to display the message, but not want to play the sound. To do that, just don't define the GoodAuthWav environment variable. Or you may not want to display a message or play a sound. In this case you would also use CounterPoint's existing configuration setting *Setup>Point of Sale>Stores>Configuration options>"Miscellaneous options">"11. Suppress messages/dialogs for: Successful card/check authorization"*. Between these two changes even a busy operator should be able to identify when an abnormal situation arises.

Because CounterPoint utilizes a Windows® API to play waves files playing wave files only works on a computer running a Windows® operating system. If you're running CounterPoint through a terminal emulator you still might not be able to hear the sound at the register. It might play on the server in the closet. It depends on your setup.

To set up a wave file for each situation you need to create an environment variable for each. Each variable will have a specific name for the authorization result and a value providing the name of the wave file to play. You can set these values in your synrun.syn file. For example, on my machine I have the values:

```
BadAuthWav=ErrSound.wav
```

```
GoodAuthWav=tada.wav
```

```
PartAuthWav=sound108.wav
```

The file names can be up to 50 characters in length. If you don't include a path in the filename the files will need to be in your CounterPoint top-level directory.

If you want to quickly identify what values a system may have assigned these environment variables go to [System>Utilities>Environment](#) and press <F1> for "Advanced settings". Towards the bottom of the list you'll find all three displayed.

## Securing Existing Credit Card Information

### *Mask the information*

If you navigate to *File Utilities>Special>Sales History* you'll find "Mask credit card numbers". This is a program that comes with CounterPoint. We didn't write it. It's source code header reads "THIS UTILITY PROGRAM PROCESSES TICKET AND PAYMENT HISTORY FILES, AND MASKS ALL... BUT THE FIRST 6 AND LAST 4 CHARACTERS OF THE CREDIT CARD NUMBERS". It also wipes out the expiration dates replacing them with "\*\*\*\*". This is a destructive action in that you lose information. But otherwise you have credit card information on file. Now this information is encrypted and the methodology has passed a PCI audit so it's not all that critical that you remove it. But it's there and someone could view the unencrypted/unmasked credit card information. That's probably the highest security risk in continuing to retain the information. There really isn't any other option when it comes to your history files. You could delete your history, but history is valuable. Note that this only addresses the credit card information in your history files. It doesn't address current customer card information at all. For the rest of this little section we'll only be addressing customer cards.

### *Purge the existing customer cards*

You can delete all of your customer cards and start over. It rather depends on your business needs and trends. We provide a nifty program that will purge customer cards based on many parameters. You could purge all of your customer cards or just the cards you used with CP-Gateway. Or maybe cards that expire within a specific date range. Or all cards based on which "slot" they exist in the customer card window. If you need your customers' credit cards on file none of these options are for you. Check out [Customers>Purge>Customer cards](#) for more information. As I write this I realize that there's no way to simply purge the credit card information from the customer file. I think the best way to accomplish (as of today) is to use the "Tokenize customer cards" program followed by the purge program. "Tokenize customer cards" will move the customer card off the customer record and onto a POS Tracking file record where the purge program could delete it.

## FCC Test Client

An optional component of FreedomPay's FCC is a test client program that you can use to test your authorization process outside of CounterPoint. You wouldn't want to install this program on all of your registers (since it could be used to authorize fake transactions), but it could prove very useful to have installed on one trusted machine per store for debugging. The FCC Test Client isn't our product so you should probably refer to FreedomPay for the latest and greatest documentation for it, but we'll discuss it a bit here.

Here's what it looks like after a successful authorization.

The screenshot displays the FCC Test Client interface. At the top, there are input fields for 'NVP Server' (127.0.0.1:1012) and 'XML Server' (127.0.0.1:1011). Below these are fields for 'Interface name' (FREEDOMPAY), 'Transport Data' (NVP Transport Data), and 'Message format' (POS\_REQ). A 'Clear/New' button and a 'Submit' button are visible. The main area is split into two panes. The left pane, titled 'Object', shows a tree view with categories: '1-Required' (ChargeAmount: 123.45, InvoiceNumber: 568, RequestType: Sale, StoreId: 8606136010, TerminalId: 2854081018), '2-Common' (AllowPartial: Y, CardNumber, CardType, ExpiryDate, FloorLimit: 50, Items, IaneId: 0, RequestId, TaxAmount: 12.34, TipAmount, Token Type: 2, UseDCC, WorkstationId: 1001), and 'Misc' (Bill To City, Bill To First Name, Bill To Last Name, Bill To Postal Code, Bill To State, Bill To Street 1, Bill To Street 2, ClerkId). The right pane, titled 'Text', shows a list of fields and their values: AccountBalance: 0, ApprovalCode: 199195, ApprovedAmount: 123.45, AvsCodeRaw, CardType: credit, CashBackAmount: 0.00, ChipData: AlyCAIwAhAegAAAAxAQiglwMJUFaoAAgACaA, CvCode, CvCodeRaw, DCCAccepted: False, Decision: A, EntryMode: icc, ErrorCode: 100, ExchangeRate: 1218, MaskedCardNumber: 476173XXXXXX0010, MerchantReferenceCode: 31EF493079DF48D7, Message: APPROVED, NameOnCard: Test/Card 01, Network: True, PinVerified: False, RequestGuid: f80c8501-4225-459b-bcab-4c2d31788e05, RequestId: 01Z6FQH4D401U61SHDD02PE7B82CP3S5, Signature: iVBORw0KGgoAAAANSUgUgAAAUoAAABNCA, SignatureFormat: png, SignatureRequired: False.

If you decide to use this tool you need to be aware that if you are operating in a production environment (Susanna and Figaro are pointed to the production gateway – not the testing gateway), any transactions you run will be real. Your “tests” will show up on someone’s credit card statement. For this reason you want to control access to any machine that has this program installed on it. I certainly don’t recommend creating a shortcut on your desktop to run it. I highly recommend that you don’t play with this program unless you understand what you’re doing.

The screen is divided into left and right panes. The left pane carries the authorization request information. It is subdivided into three sections for fields that are required, commonly provided and miscellaneous. In my testing I usually enter fields in all three sections. The right pane displays the results. The upper-left corner of the screen has several fields most of which you don't care about. The only important field there (important for your purposes anyway) is the field named "NVP Server". It needs to have the IP address and port number of the machine that's running Figaro. The format of the field is the IP address and port number separated by a colon character like this "*ip-address:port-number*". In my screenshot you'll see the value "127.0.0.1:1012". The IP address "127.0.0.1" is known as the "loop-back" address and indicates "on this machine". The port number "1012" matches the "nvpPort" parameter in Figaro's configuration file. The default values for these fields are retained in the Test Client's configuration file. See the relevant section in "[Installation and Configuration](#)" for those details.

Note that each pane has tabs at the top. You'll probably never need to use anything but the default "Object" tabs. The other tabs show the same data in different formats which is more a tool for developers than end-users. But you might find them interesting if you're so inclined.

Not all of the fields to be populated to perform an authorization. Here are the fields we typically put values in when testing.

ChargeAmount	<i>(To prompt for a signature enter a value above your FloorLimit field)</i>
InvoiceNumber	<i>(whatever you want)</i>
RequestType	Sale
StoreId	<i>(See paragraph# 1 below)</i>
TerminalId	<i>(See paragraph# 1 below)</i>
AllowPartial	Y
FloorLimit	100
TaxAmount	<i>(whatever you want)</i>
TokenType	2
WorkstationId	<i>(See paragraph# 2 below)</i>
ClerkId	<i>(whatever you want)</i>
EnableAVS	Y
RegisterNumber	<i>(whatever you want)</i>
SignatureFormat	png
UpdateToken	Y

1. The StationId and TerminalId fields need to be populated with the values FreedomPay assigned you when you signed up. Otherwise it won't work.
2. Each Susanna has a workstationId parameter in its configuration file. The test client's WorkstationId fields needs to match the corresponding value in the Susanna that is controlling the desired Ingenico. Otherwise Figaro will send the authorization request to the wrong Ingenico (or no Ingenico at all). If you attempt to run a test and your local Ingenico never wakes up, but you receive a call from another register (possibly in another store) that their Ingenico is prompting for a card when there's no transaction happening there, you can be sure this is the problem.

The Test Client doesn't retain the values of any of these fields. Each time you close the program any values you entered are thrown away. So make certain you're done with the program before you close it lest you'll have to rekey all those values again.

Click the "Submit" button and wait for the Ingenico to wake up. Follow the prompts on the Ingenico then watch for results to appear in the right hand pane. (Transactions that don't require a card such as voids won't involve the Ingenico.) Look in the right-side pane for fields named "Decision" and "ErrorCode". They should have the values "A" and "100" respectively. If so, the field "ApprovalCode" should have a value and the field "Message" should have the value "APPROVED".

The default value for the request field "RequestType" is "Sale". That's generally what you want to use. But it allows other values. One is "Refund" which you might want to use to cancel any Sale transaction you run especially if you're testing in a production environment. Just change RequestType from "Sale" to "Refund" and click Submit again. You could use "Void" to make it appear that the original sale transaction never happened, but to make that work you'd need to provide the original transaction's request-id. That's a bit geeky. Another value is "TruncateDatabase". It is powerful and dangerous and I'm hesitant to even mention it. But we've had to use before and you might, but only as a last resort. So let's go there.

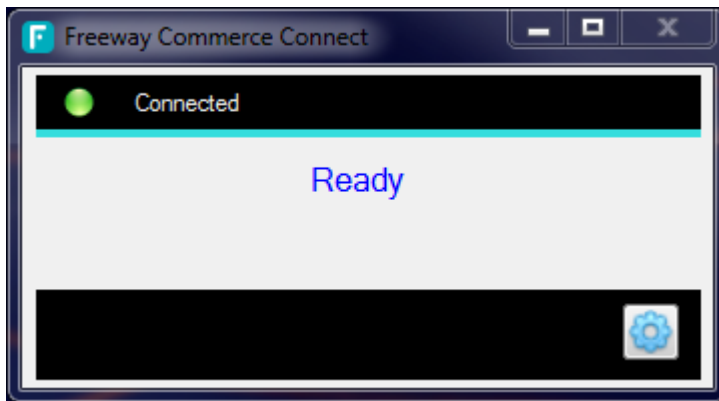
Figaro has its own database with several tables. The TruncateDatabase command initializes all of the tables in Figaro's database essentially wiping them out. One of these is the table that retains offline transactions that are being retried. If you have unapproved authorization requests in Figaro's database when you execute a TruncateDatabase command these requests will be erased from the database and Figaro will no longer retry them. Thus you will have issued an offline approval for a transaction you'll never be paid for. If you configure FCC not to perform offline authorizations this won't be a problem. Generally speaking, I recommend that you just forget about TruncateDatabase.

So why do I tell you at all ? Mostly to prevent you from trying it out. With a single "What's this do ?" you could lose thousands of dollars of pending offline transactions. The other reason is more real world. During our combined development cycle we created invalid authorization transactions that Figaro categorized as offline. Since they were invalid they would never authorize and so were stuck in Figaro. Because of the default values in our Figaro configuration file these transactions would try to authorize every 5 seconds. Since we had quite a few of these transactions trying to authorize over and over Figaro's log file grew in size rapidly. Since FreedomPay has no way to selectively delete transactions from this table TruncateDatabase was the only way to clear the transactions from Figaro's database. (FreedomPay is hard at work adding the code to better handle this process.) In a post-development environment you shouldn't have such a problem, but if such a transaction were to occur, as of the time of this document, this is the only way to handle it.

## FCC Client User Interface (FCC-UI)

An optional component of FreedomPay's FCC is a client-side user interface that displays the status of the authorization process to the clerk. This allows the clerk to follow what's happening without having to watch the customer. It's a small window that pops up in the lower right corner of the screen when the authorization process begins. (Starting with FCC v4.x you can reposition the window anywhere on the screen.) The text it displays changes as the authorization process takes place. It is not required. You can decide not to install it, but we recommend that you do. It has its own configuration file which is discussed in the section "Installation and configuration" below.

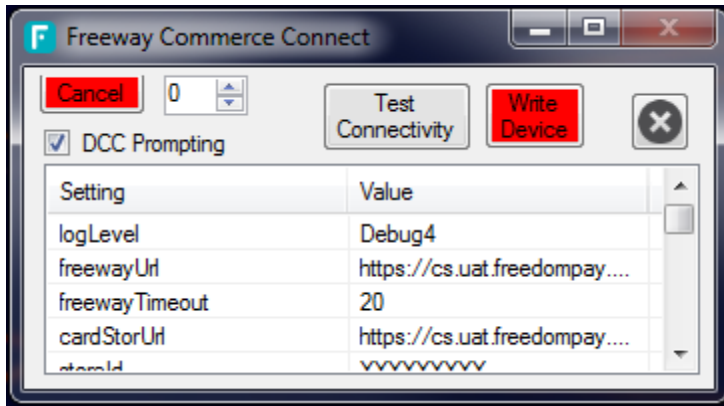
Here is what it looks like.



The FCC-UI is not a Windows service like Figaro and Susanna. It runs like a typical Windows program, but "lives" in the System Tray in the lower-right corner of the screen. If it is running you should see an icon with a white "F" on a blue background. If you single-left click on the icon it will display on the screen for 60 seconds then hide itself. If the icon is missing from the System Tray the program isn't running. To start it, use Windows Explorer to navigate to the executables directory ("C:\Program Files (x86)\FreedomPay\FreewayCommerceConnect") and double-left-click on "FCCUI.exe". The FCC-UI icon will appear on the System Tray. Single left-click it and it will open somewhere on your screen. You can position it wherever you want it. It will remember this position and stay there each time it starts in the future.

Note that if you look for the FCC-UI in Windows Task Manager's Applications tab it won't be there unless you've clicked on it and brought it onto the screen. There it has the ambiguous (and user-configurable) name "Freeway Commerce Connect". It always displays (if it's running of course) on the Processes tab under the name "FCCUI.exe". So if you want to stop it for some reason that's where it is.

The FCC-UI is more than a display tool. It allows testing the connection to Figaro, Susanna and the Ingenico. It also allows updating the Ingenico with the latest firmware, configuration file and advertising with a single click (albeit via the obsolete FileWrite update process). However there is some set up required for this work. That process is described later in this document under [FileWrite](#). By clicking on the gear icon in the lower-right of the window the interface will change to maintenance mode like this.



The window isn't labeled particularly well, but there are some key elements here. The "Test Connectivity" button sends a "create token" command to Susanna to fulfill. If everything is configured correctly you'll get a good result back. In fact you might get an error back (like I do) and yet the test proved that things are configured correctly. It depends on the error you get back. My error is that the token cannot be created. But that's correct on my system since I'm not configured to create tokens. But the underlying communications did in fact test OK.

The "Cancel" button stops a currently executing test.

The "Write Device" button performs an immediate automatic update. The program bypasses the automatic update settings and goes through the normal chain of events to locate and install any updates for the attached Ingenico. This is very convenient when you first set up your Ingenico. But it could be clicked on by pretty much anyone after that as well. So FreedomPay provides an FCC-UI parameter

```
<add key="showFileWriteButton" value="true" />
```

to control whether the "Write Device" button appears on the screen. You may want to set up all of your Ingenicos then change this setting to "false" to prevent your clerks from updating individual Ingenicos on-the-fly.

The greyed-out "X" button ends maintenance mode and returns FCC-UI to monitoring mode.

There's not a lot in the FCC-UI configuration file that you can change. You can change the window's title to something more specific. There's the "showWriteFileButton" parameter. There are a few parameters that control how long various messages stay on the screen. See the section for the FCC-UI in "Installation and Configuration" below for details.

If you make a change to FCC-UI's configuration file you'll need to restart it for the changes to take effect. You'll quickly find that that once FCC-UI is started it can't be stopped. There's no "Exit" menu selection or button to click to make that happen. That's by design. You need to use Windows Task Manager to stop it then start it again as described above.



# Installation and Configuration

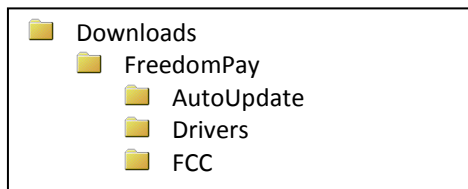
## Freeway Commerce Connect (FCC)

You can download supported versions of FreedomPay's FCC from my DropBox. I'll send you a share link upon request.

My DropBox pages contain several versions of the FCC and CP-EMV. It also carries documentation sample configuration files and drivers for Ingenicos. Navigating around should be pretty self-explanatory.

Each version of the FCC has its own folder. Newer versions contain a single installation file named such as FCC\_5.0.13.2.zip. This file is the "Full" installation set which includes both the Figaro and Susanna software. This file is almost 100meg. It doesn't contain the requisite minimum version 4.6.2 of the Microsoft .NET runtime. However that is downloaded automatically by the installer if it is missing. Older versions of the FCC have two zip files containing the software. But this document is tailored for version 5.0x so we're going to skip discussing the older version's packaging. Refer to the older version's documentation for that.

There are going to be a few directories to keep organized so it's best to start with good practices up front. We recommend creating a directory structure that will allow you to keep organized with the many possible files you'll need to properly install and configure the various bits of software and hardware possible with FCC. For example, we created the directories:



We'll cover the use of these directories as we proceed through this document. It will make sense by the end. Notice there's only a single directory for the FCC. That's because each version of the downloaded file has its version numbers embedded in it. So that's easy enough to understand to throw all of the various versions together in a single directory.

## USB Drivers for the Ingenico

Before you can successfully connect your Ingenico to your computer you might need to install the driver that allows communications between the two. A driver is a set of routines that allow an operating system to communicate with some specific piece of hardware. FCC is designed to access the Ingenico via their specific driver. It's not uncommon for a Windows update to replace the Ingenico driver with one that doesn't work. If your Ingenico stops connecting when it used to work fine before, examine the computers' update history. I bet you'll find the computer updated recently. Just reinstall the driver.

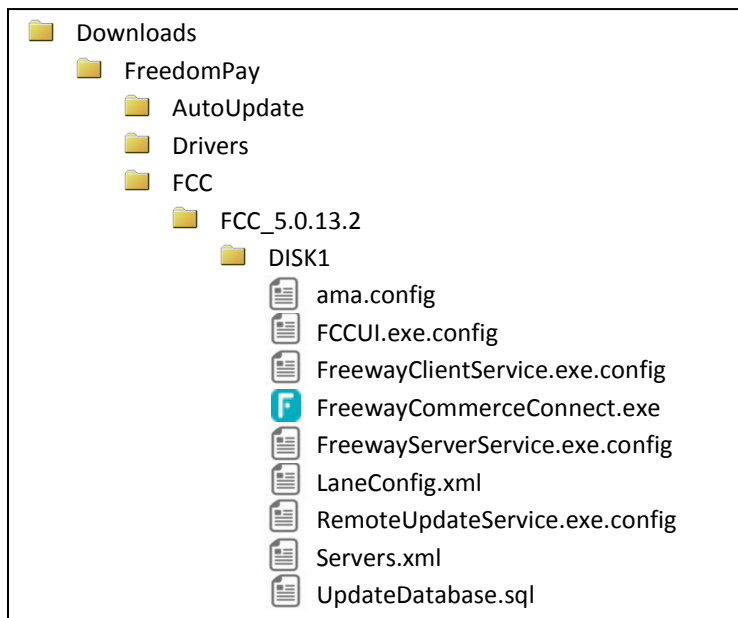
You can find the driver on my DropBox. I'll send you a share link upon request. As of the date I am writing this I have three drivers in my DropBox. The latest version is named Driver\_v1.2.8.1.zip. Download the file into C:\Downloads\FreedomPay\Drivers (if you're using my suggested directory structure). Unzip the file. You'll find three files for three different platforms.

FreewayMsrDriverMsi.msi	For 64bit installations of Windows Vista and higher
FreewayMsrDriverMsi_x86.msi	For 32bit installations of Windows Vista and higher
FreewayMsrDriverMsi_XP.msi	For Windows XP only

Just double-click on the file and the installation process will kick off. There are no options or parameters. You just need to acknowledge that you want to do it and accept the defaults.

## Freeway Commerce Connect Components

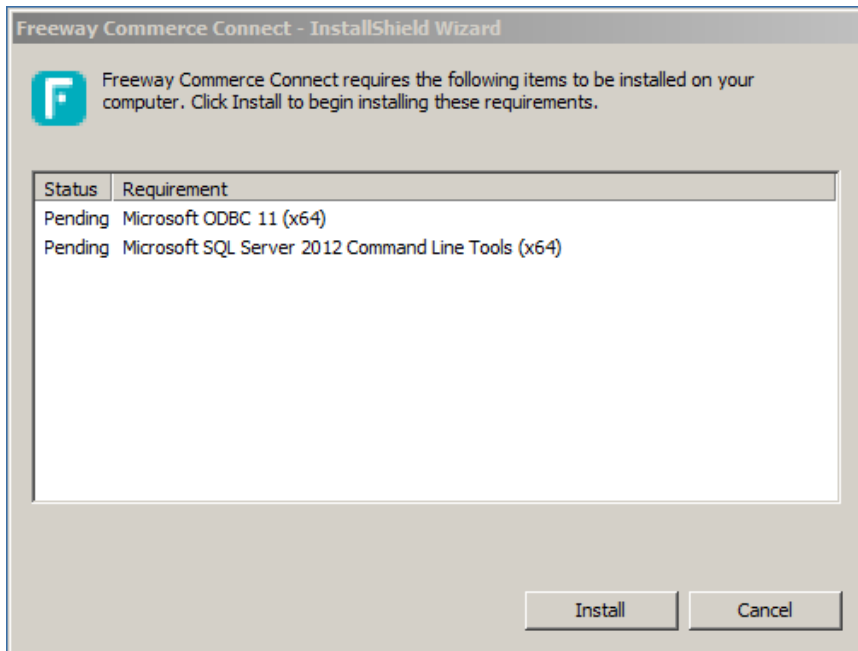
Let's assume you downloaded the FCC installation file into C:\Downloads\FreedomPay\FCC as in my example. Extract the contents of the file preserving the directory structure within it. After extraction you'll find:



FreewayCommerceConnect.exe is the program that you'll run to start the installation. But first take note of the files FreewayClientService.exe.config, FreewayServerService.exe.config and severs.xml. The first two are default configuration files for Susanna and Figaro. The third provides the network address for Figaro. If this is the first time you've installed FCC and you don't already have configuration files to use leave these files as they are and start the installation. However, if you already have configuration files and you want the installation process to copy and use those files, replace the default files in the DISK1 directory with your files. Now you're ready to start the installation.

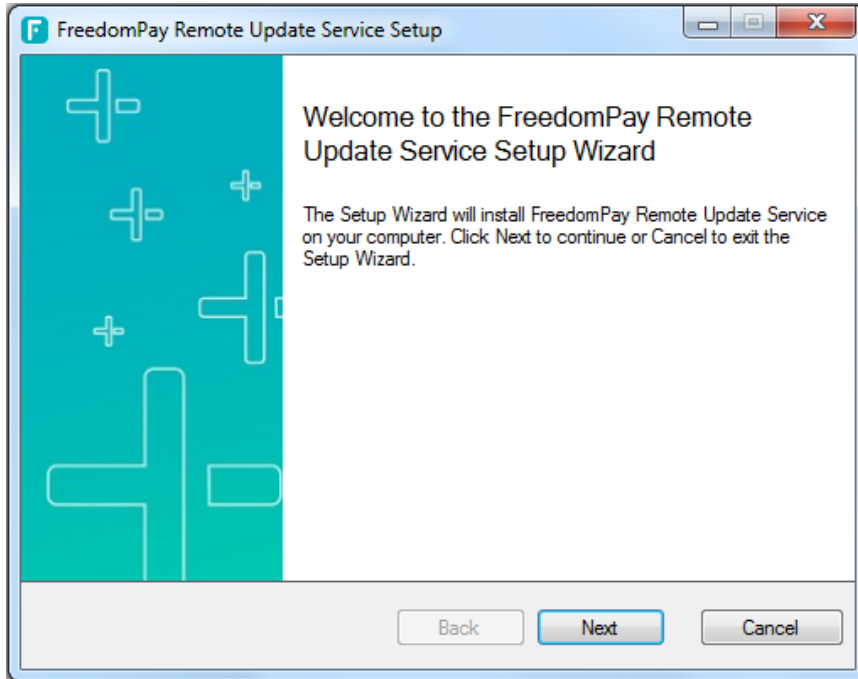
### *Installing FCC Interactively*

Right-click on FreewayCommerceConnect.exe and select "Run as administrator". You'll get a warning screen asking "Do you want to allow the following program from an unknown publisher to make changes to this computer?". Click "Yes". If your system is missing a prerequisite piece of software you'll be presented with a screen that looks something like this.

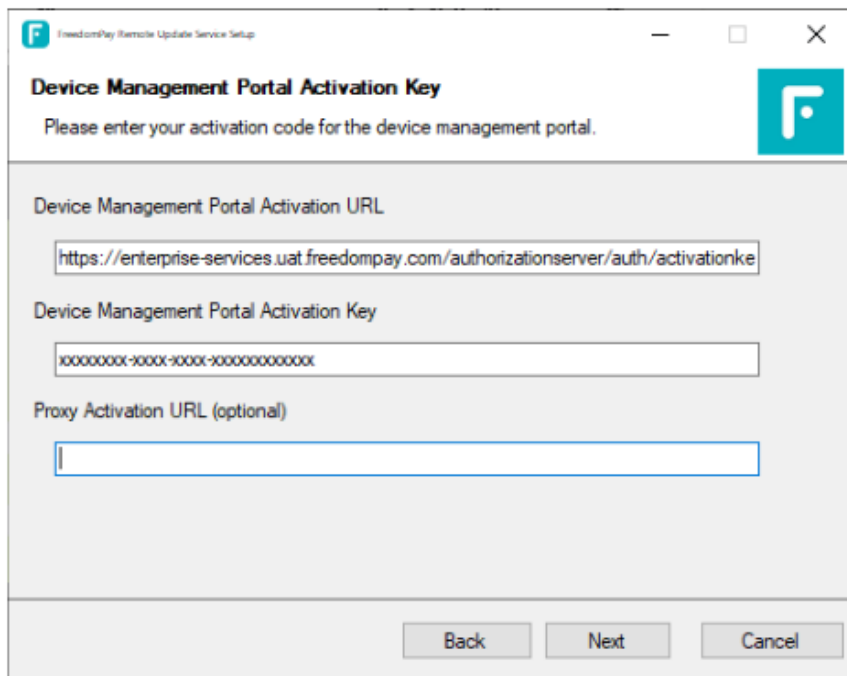


You'll need to click "Install" to install the missing components before you can continue. Each missing component has its own installation routine. So you'll probably have to wade through a splash screen, a license acceptance screen, a features selection screen, a last chance screen, an "Installing" screen and finally an "Installation complete" screen for each component. Just accept the default values and click "Next", "Install" or "Finish" on each until they're installed. If you do install prerequisites you should probably reboot before continuing the installation.

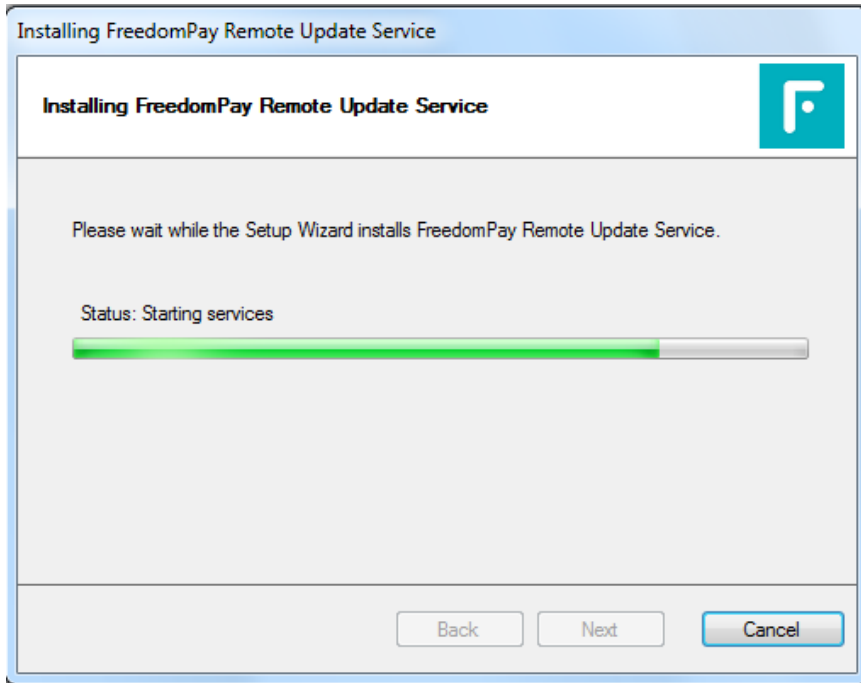
Once the missing components are installed the installation continues with the ubiquitous “Welcome” splash screen, but not for the FCC – for the Remote Update Service Wizard.



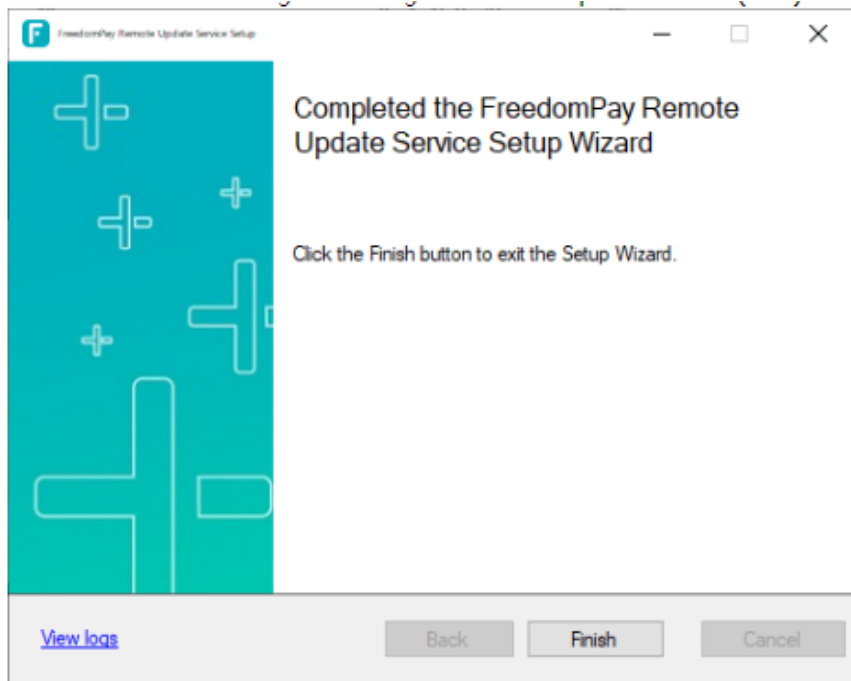
Click “Next”. The installation will begin. It will prompt you for a Device Management Portal key. If you don’t have one you’re already done - because it is required ! You’ll need to get your key from your FreedomPay Rep before you can continue. If you have it, key it in.



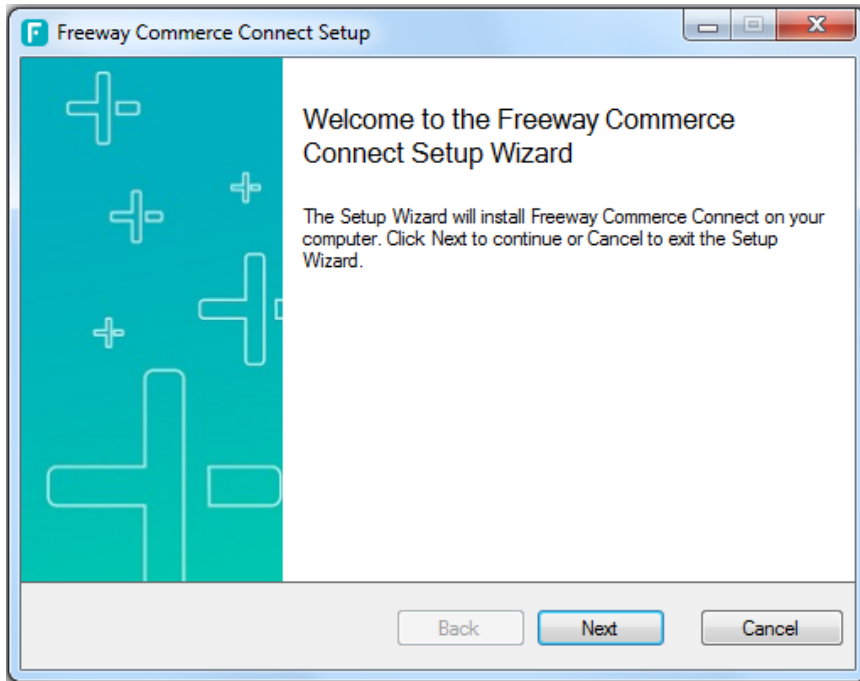
The installation continues until...



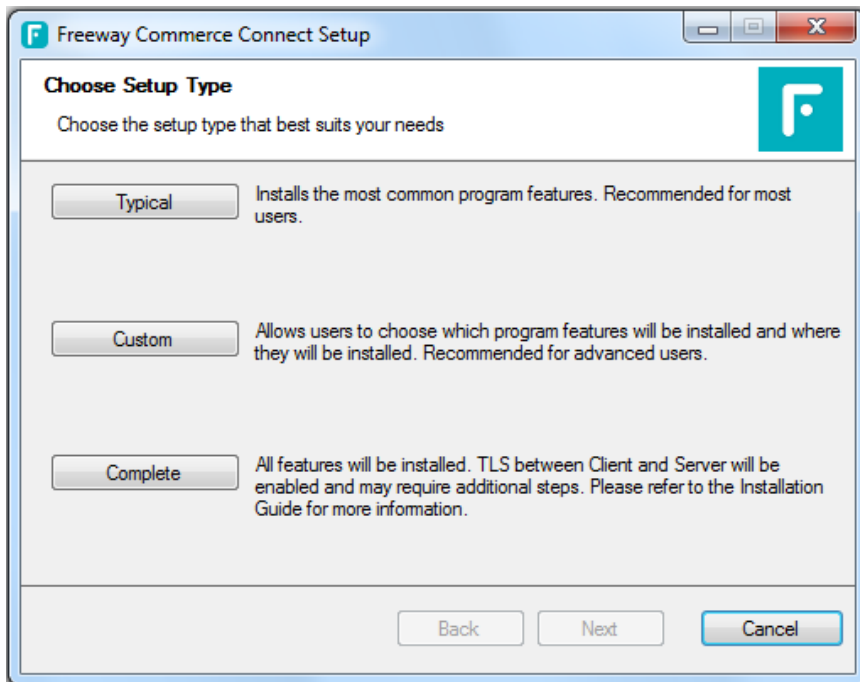
Once the installation is complete you'll get this



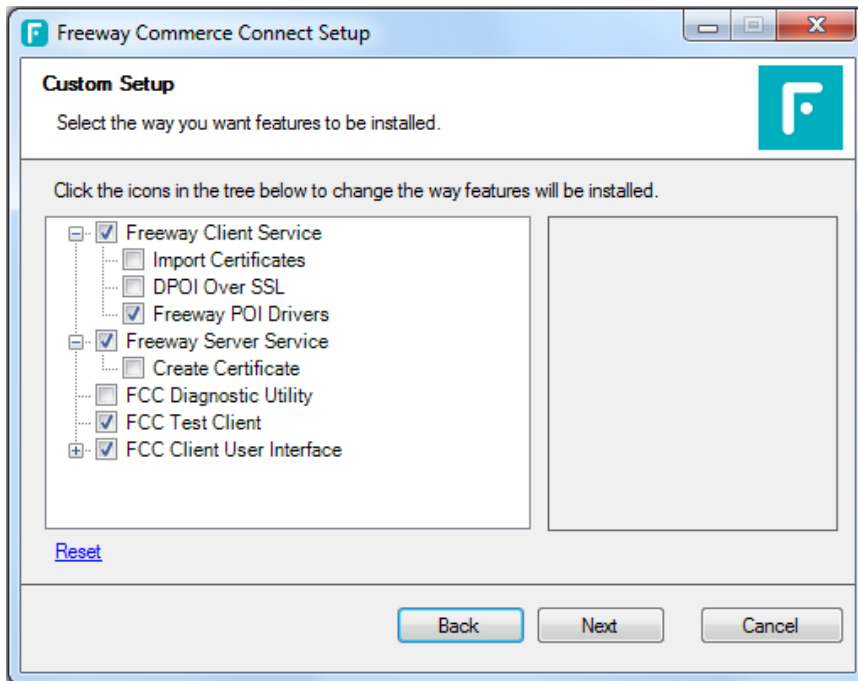
Now (finally) you arrive at the FCC “Welcome” Splash screen.



Right off the bat you need to make a decision. “Typical” install Figaro and Susanna on the machine which probably isn’t right for you. That assumes you intend to run multiple Figaros (one on each register) which will only work if you have a single register. Most likely you will install “Custom” to install either Figaro (and its components) or Susanna (and its components including the FCCUI).

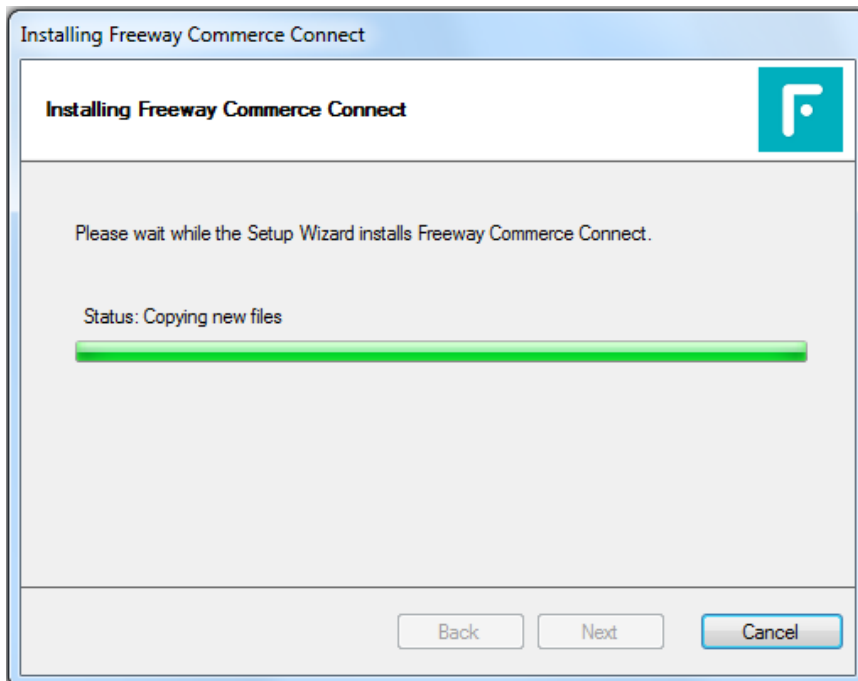


So we'll assume you click "Custom". That presents you this screen.

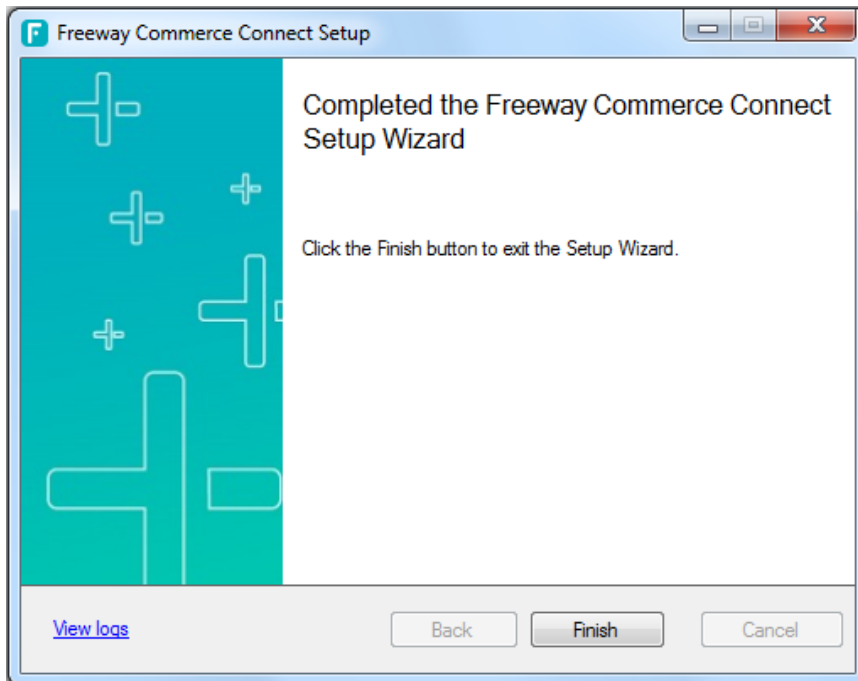


If you're installing Susanna keep "Freeway Client Service", "Freeway POI Drivers" and "FCC Client User Interface" checked and clear the rest. If you're installing Figaro check "Freeway Server Service" and optionally "FCC Test Client". Clear the rest. If you're installing Figaro and Susanna on the same machine (on purpose) leave the selections as they are.

Click "Next". The installation will begin copying files and eventually will start the Figaro and Susanna services.



Eventually you'll get this !



Click "Finish".

But you're not done. Unless you are performing an upgrade (and you substituted your own configuration files for the default configuration files) you'll need to edit the configuration files for the components you installed. The configuration files for "FCC Client User Interface" and "FCC Test Client" are probably OK as they were initially delivered. But the configuration files for Figaro and Susanna will definitely need tweaking. More on that in a bit.

### *Installing FCC From the Windows Command Line*

The FCC installation program can also be run from the command line in a non-interactive mode. This is a powerful feature that lends itself to automating much of the monotony of installing Susannas on many registers. Add these features together:

- The installation can be run from a "Batch" file
- It can be configured to install specific FCC components
- The installation process can pass user-defined configuration files to the executable directory

That means the installation command can be placed in a batch file with other commands to perform functions prior to and after the installation process such as retrieve configuration files from a central location, tweaking values prior to installation, prompting for additional information (such as a workstationID) and restarting the service.



The syntax for running the FCC installation from the command line is:

```
FreewayCommerceConnect.exe /s /v"/quiet ADDLOCAL="components-to-install" /L*v fccinstall.log"
```

Admittedly that font is pretty small. But I wanted to ensure I got the whole command on a single line. The parameter "*components-to-install*" can be one or a combination of the following values:

CreateDatabase	Create the FCC Server Service (Figaro) database
CreateDbDir	Create the FCC ProgramData directories
FCCClient	Install the FCC Client Service (Susanna) program
FCCClientConfig	Install the FCC Client Service (Susanna) configuration file only
FCCUI	Install the FCC User Interface program
FCCUIConfig	Install the FCC User Interface configuration file only
FCCUI_XP	Install the FCC User Interface program (Windows XP SP3 only)
FCCUI_XPConfig	Install the FCC User Interface configuration file only (Windows XP SP3 only)
FCCServer	Install the FCC Server Service (Figaro) program
FCCServerConfig	Install the FCC Server Service (Figaro) configuration file only
FCCTestClient	Install the FCC Test Client program

Need I say that if you use this capability you need to put some thought into the combination of values for "components-to-install". If you intend to install Figaro don't specify "FCCServer" and "FCCClientConfig" because that's apples-to-oranges.

The first "must do" rule is if you are installing FCC for the first time is you must include "CreateDbDir". This directs the installer to create FCC's ProgramData directory structures. These directories are where FCC creates Figaro's database and writes its log files. The only exception to this is if you are installing just the FCC User Interface. That program doesn't use either the database or create a log file.

The commands are pretty long and it's impossible to put them on this page in a font that allows the command to appear on a single line AND be legible. So the following page is going to be formatted in landscape merely to fit the sample commands on the page in a legible font size.

Generally speaking, if you install Figaro, you'll install its configuration file and create its database. That command is going to look like this:

```
FreewayCommerceConnect.exe /s /v"/quiet ADDLOCAL="CreateDbDir,FCCServer,CreateDatabase,FCCServerConfig" /L*v fccinstall.log"
```

Notice the command includes the following parameters:

CreateDatabase	Create the FCC Server Service (Figaro) database
CreateDbDir	Create the FCC ProgramData directories
FCCServer	Install the FCC Server Service (Figaro) program
FCCServerConfig	Install the FCC Server Service (Figaro) configuration file only

If you install Susanna, you'll indicate its configuration file and the User Interface as well like this:

```
FreewayCommerceConnect.exe /s /v"/quiet ADDLOCAL="CreateDbDir,FCCClient,FCCClientConfig,FCCUI,FCCUIConfig" /L*v fccinstall.log"
```

This command includes the following parameters:

CreateDbDir	Create the FCC ProgramData directories
FCCClient	Install the FCC Client Service (Susanna) program
FCCClientConfig	Install the FCC Client Service (Susanna) configuration file only
FCCUI	Install the FCC User Interface program
FCCUIConfig	Install the FCC User Interface configuration file only

If you want to install the FCC Test Client and User Interface on a separate, secured machine you'd use this command:

```
FreewayCommerceConnect.exe /s /v"/quiet ADDLOCAL="CreateDbDir,FCCTestClient,FCCUI,FCCUIConfig " /L*v fccinstall.log"
```

CreateDbDir	Create the FCC ProgramData directories
FCCTestClient	Install the FCC Test Client program

To uninstall FreedomPay software use the command:

```
msiexec.exe /x{28FEB192-EE11-49AF-A6E8-812799CE9BF8} /quiet /L*v FCCUninstallLog.log
```

When you first saw the sample commands you probably thought “What a command line ! How am I supposed to type all that without a single mistake ?” I suggest you cut the samples from this document and paste them into 3 batch files named InstallFccServer.bat, InstallFccClient.bat and installFccTestClient.bat. You can use any names you want, but even these names are shorter and more clear than typing the commands. Having the batch files will now allow you to add additional operating system commands before and after the install.

So let’s assume you created batch files with the names mentioned above. Create (or copy) them into the DISK1 directory so you won’t have any PATH environment variable issues.

Because the installation process makes changes to the computer it needs to be run as an administrator. Open a Windows Command Prompt in administrator mode by single right-clicking the menu selection and single left-clicking “Run as administrator”. You’ll be presented the warning prompt “Do you want to allow the following program to make changes to this computer”. Click “Yes”. When the command line session opens it should have the title “Administrator: Command Prompt”.

Use the CD command to navigate to the DISK1 directory. To run the desired installation just type the name of the batch file and press <ENTER>. Since you’re already running the session as an administrator you shouldn’t receive any warnings from Windows about making changes to the computer or copying files with varying permissions.

### *Susanna Configuration File Upgrade Utilities*

Looking at the aforementioned DISK1 directory you’ll see there are copies of the Figaro and Susanna configuration files. That’s not an accident. During the installation process these files are copied into the FreedomPay executable directory and so become the configuration files used by both services. This is especially handy if you have a lot of Susannas to install (or upgrade). The idea is you create a template configuration file with all the desired settings (except for a couple we’ll get to), copy the file into the DISK1 directory and zip up the installation set for use by your entire enterprise. Having a single standardized installation set for all of your stores will help you maintain your Figaros and Susannas in a known configuration which makes support that much easier.

Most likely all of your Figaro installations will use the exact same configuration file. Our sample Figaro configuration file (found on our DropBox) will probably be correct for you as-is with no modifications at all. So from this point forward we’re going to be discussing upgrading the Susanna service.

Now, about those two settings. The first is the workstation-id that identifies the Susanna. Each register has a unique value for this setting so no one configuration file can work for all (or even two) registers. This setting must be made unique before the installation is complete. We provide two utilities to simplify the situation for upgrades. Both utilities attempt to retrieve the workstation-id from the existing Susanna configuration file and update the template configuration file before the installation program is invoked. To employ either utility copy the program(s) to the DISK1 directory and use one of the scripts we provide (UpgradeClient.bat and MergeClient.bat) to drive the upgrade process. We’ll discuss these scripts after we’ve discussed the utility programs.

## UpgradeConfig.exe

Use this program to upgrade an existing v4.x Susanna to v5.x. In version 4.x the Susanna configuration file carried the register's workstation-id in an XML field named "workstationID". In the version 5.x configuration file this field was split into two fields – "workstationIdMethod" and "workstationIdMethodData". The setting for "workstationIdMethod" will be the same value for all of your registers ("ConfigFile") so that value should already be set in your template configuration file (as it already is in our sample Susanna configuration file). It's the "workstationIdMethodData" setting that needs to be changed for each register.

To launch the program just invoke the name of the executable – UpgradeConfig.exe. If the program runs to completion with no errors it will set ERRORLEVEL to "0" (zero). If an error is encountered it will set ERRORLEVEL to a value that is unique to the error encountered. If you want to see a list of the possible ERRORLEVEL values launch the program with the parameter "/?" like this:

```
UpgradeConfig.exe /?
```

This will display a "Help" screen listing them all. The UpdateConfig.bat file we provide checks ERRORLEVEL after the program is run and acts accordingly.

As the program runs it checks that both configuration files exist (v4.x and v5.x) and both have the expected settings. It expects the existing 4.x Susanna configuration file to be in the default directory. So if you changed that during the installation neither of these utilities will work for you. It expects the v5.x configuration file (the new template configuration file) to be in the same directory as it is. It expects the v5.x setting "workstationIdMethodData" to have the value "TERMINAL1". When the program runs it will replace "TERMINAL1" with the value of the "workstationID" setting currently in the v4.x configuration file. If "TERMINAL1" is missing the program won't find where to put the setting.

The other configuration setting this utility upgrades is named "quickServeFloorLimit". As discussed before this setting carries the threshold authorization amount that would require a customer's signature. This setting probably doesn't change from register to register, but it might be different for every store in your enterprise. The v4.x and v5.x settings have the same name and so this process is pretty straightforward. However the template v5.x Susanna configuration file needs to have the value "50" for this program to find and update its value. Our sample Susanna configuration file has this value already.

If the program runs to completion with no errors it saves the template v5.x Susanna configuration file as "FreeWayClientService.exe.config.saved" and the upgraded configuration file with the expected name "FreeWayClientService.exe.config".

## MergeConfig.exe

Use this program to update an existing v5.x Susanna. Since the major version numbers are the same both configuration files contain settings with the same names – “workstationIdMethodData” and “quickServeFloorLimit”. Like UpgradeConfig this program expects the template Susanna configuration file to have the value “TERMINAL1” for “workstationIdMethodData” and “50” for “quickServeFloorLimit”.

To launch the program just invoke the name of the executable – MergeConfig.exe. If the program runs to completion with no errors it will set ERRORLEVEL to “0” (zero). If an error is encountered it will set ERRORLEVEL to a value that is unique to the error encountered. If you want to see a list of the possible ERRORLEVEL values launch the program with the parameter “/?” like this:

```
MergeConfig.exe /?
```

This will display a “Help” screen listing them all. The MergeConfig.bat file we provide checks ERRORLEVEL after the program is run and acts accordingly.

As the program runs it checks that both configuration files exist and both have the expected settings. It expects the existing v5.x Susanna configuration file to be in the default directory. So if you changed that during the installation neither of these utilities will work for you. It expects the v5.x template configuration file to be in the same directory as it is. It expects the v5.x setting “workstationIdMethodData” to have the value “TERMINAL1”. When the program runs it will replace “TERMINAL1” with the value of the same setting the current v5.x configuration file. If “TERMINAL1” is missing the program won’t find where to put the setting.

The other configuration setting this utility upgrades is named “quickServeFloorLimit”. As discussed before this setting carries the threshold authorization amount that would require a customer’s signature. This setting probably doesn’t change from register to register, but it might be different for every store in your enterprise. The template and current settings have the same name and so this process is pretty straightforward. However the template v5.x Susanna configuration file needs to have the value “50” for this program to find and update its value. Our sample Susanna configuration file has this value already.

If the program runs to completion with no errors it saves the template v5.x Susanna configuration file as “FreeWayClientService.exe.config.saved” and the upgraded configuration file with the expected name “FreeWayClientService.exe.config”.

## UpgradeClient.bat

Use this script to upgrade an existing v4.x Susanna to version 5.x. Despite the considerable length of this script its heart and purpose is to invoke UpgradeConfig.exe to upgrade the template v5.x Susanna configuration file. Because it starts and starts the Susanna service it must be run as an administrator. The best way to accomplish this is to open a command line session as an administrator (by right clicking and selecting “Run as administrator”), navigate to the DISK1 directory and invoke the script from there. This is important because the first check the script makes is to ensure it is being run as an administrator.

The script does a lot. You can open it in an editor and check it out if you’re into that sort of thing. I’ll just bullet list the steps below. When I say the “current” directory I mean the DISK1 directory where all of these files are supposed to reside. When I say the “FCC” directory I mean the FreedomPay executable directory where the v4.x Susanna has already been installed.

1. Checks it’s being run in administrator mode.
2. Checks if it has already been run.
3. Determines the FCC executable directory.
4. Checks that servers.xml exists in the current directory.
5. Checks that FCCUI.exe.config exists in the current directory.
6. Checks that FreeWayClientService.exe.config exists in the current directory.
7. Checks that servers.xml exists in the FCC directory.
8. Checks that FreeWayClientService.exe.config exists in the FCC directory.
9. Checks that UpgradeConfig.exe exists in the current directory.
10. Checks that FreewayCommerceConnect.exe is in the current directory.
11. Invokes UpgradeConfig.exe and exits if there’s an error.
12. Copy the updated template configuration to the FCC directory.
13. Copy servers.xml from the current directory to the FCC directory.
14. Copy FCCUI.exe.config from the current directory to the FCC directory.
15. Invoke the FreedomPay installation program.
16. Copy the updated template configuration to the FCC directory.
17. If UpgraderBA.bat exists run it.
18. Stops then starts the Susanna service.

Yeah... That’s a lot.

If you read all that you might have noticed references to servers.xml and FCCUI.exe.config. servers.xml “points” to Figaro and is likely the same for every register in a store and possibly your entire enterprise. It is unlikely that you’ve modified FCCUI.exe.config, but some people have so we handle that file as well. We have found the installation process will stomp on these two files (and others) so we copy it from the FreedomPay executable directory before starting the installation program. In fact you might notice the script copies some files more than once.

Then in step 17 it calls another script named UpgraderBA.bat. This script sets up PAL to push an update into the attached Ingenico. UpgraderBA.bat doesn’t need to exist to be called. If you omit it from the DISK1 directory UpgradeClient.bat will continue running. Once everything is done the script restarts Susanna to put all of the changes into effect.

## UpdateClient.bat

Use this script to update an existing v5.x Susanna. Despite the considerable length of this script its heart and purpose is to invoke MergeConfig.exe to upgrade the template v5.x Susanna configuration file. Because it starts and starts the Susanna service it must be run as an administrator. The best way to accomplish this is to open a command line session as an administrator (by right clicking and selecting “Run as administrator”), navigate to the DISK1 directory and invoke the script from there. This is important because the first check the script makes is to ensure it is being run as an administrator.

The script does a lot. You can open it in an editor and check it out if you’re into that sort of thing. I’ll just bullet list the steps below. When I say the “current” directory I mean the DISK1 directory where all of these files are supposed to reside. When I say the “FCC” directory I mean the FreedomPay executable directory where the v5.x Susanna has already been installed.

1. Checks it’s being run in administrator mode.
2. Checks if it has already been run.
3. Determines the FCC executable directory.
4. Checks that servers.xml exists in the current directory.
5. Checks that FCCUI.exe.config exists in the current directory.
6. Checks that FreeWayClientService.exe.config exists in the current directory.
7. Checks that servers.xml exists in the FCC directory.
8. Checks that FreeWayClientService.exe.config exists in the FCC directory.
9. Checks that MergeConfig.exe exists in the current directory.
10. Checks that FreewayCommerceConnect.exe is in the current directory.
11. Invokes MergeConfig.exe and exits if there’s an error.
12. Copy the updated template configuration to the FCC directory.
13. Copy servers.xml from the current directory to the FCC directory.
14. Copy FCCUI.exe.config from the current directory to the FCC directory.
15. Invoke the FreedomPay installation program.
16. Copy the updated template configuration to the FCC directory.
17. If UpgraderRBA.bat exists run it.
18. Stops then starts the Susanna service.

Yeah... That’s a lot.

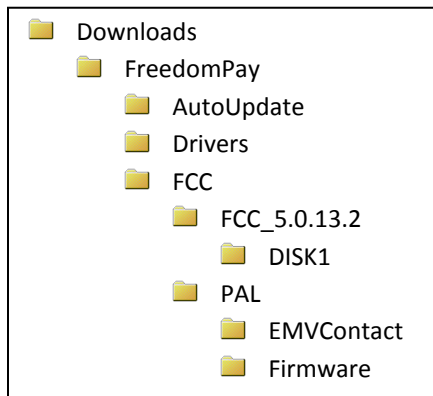
If you read all that you might have noticed references to servers.xml and FCCUI.exe.config. servers.xml “points” to Figaro and is likely the same for every register in a store and possibly your entire enterprise. It is unlikely that you’ve modified FCCUI.exe.config, but some people have so we handle that file as well. We have found the installation process will stomp on these two files (and others) so we copy it from the FreedomPay executable directory before starting the installation program. In fact you might notice the script copies some files more than once.

Then in step 17 it calls another script named UpgraderRBA.bat. This script sets up PAL to push an update into the attached Ingenico. UpgraderRBA.bat doesn’t need to exist to be called. If you omit it from the DISK1 directory UpgradeClient.bat will continue running. Once everything is done the script restarts Susanna to put all of the changes into effect.

## UpgradeRBA.bat

You might have occasion to update the software on an Ingenico. Maybe you've decided to add a new capability like debit cards or contactless. Or you've replaced an Ingenico with an older version of the RBA and you need to update it. Or maybe you're updating the version of FreedomPay's FCC suite and you need to update the RBAs running on all of your Ingenicos to match. This last situation is why we added UpgradeRBA.bat to UpdateClient.bat and UpgradeClient.bat.

This script basically copies the directory structure from the installation set into the PAL directories. Then the next time Susanna is restarted PAL uses the copied files to update the attached Ingenico. The directory structure looks like this:



The PAL directory is one level “higher” than the DISK1 directory.

Here are the bullet points of what it does.

1. Checks that it is being run in administrator mode.
2. Has the operator confirm they want to continue.
3. Determines the current PAL directory .
4. Copies the files from ..\PAL\\*. \* to the current PAL directory.
5. Ask the operator to restart Susanna or not.
6. Stop then start Susanna if selected.



## Post-Installation Steps

All of the executable files associated with FCC will be installed in the directory “C:\Program Files (x86)\FreedomPay\FreewayCommerceConnect” (Windows 7). For convenience we call this the “executables” directory. You may notice this directory also has some other directories named “localUpdateDir”, “log” and “manifestsRun”. We’ll cover those later.

There are four executables in this directory that will concern you. Each of them has a configuration file associated with them. Plus there are a couple of loose configuration files. Here they are:

FreewayServerService.exe	FCC Server Service (Figaro)
FreewayServerService.exe.config	FCC Server Service configuration file
FreewayClientService.exe	FCC Client Service (Susanna)
FreewayClientService.exe.config	FCC Client Service configuration file
FCCTestClient.exe	FCC Test client
FCCTestClient.exe.config	FCC Test client’s configuration file
FCCUI.exe	FCC User interface
FCCUI.exe.config	FCC User interface’s configuration file
servers.xml	File containing Figaro’s IP address / URL

Each of these files will be discussed in the sections below.

Before you make any changes to an original configuration file you should make a backup copy of it. Be proactive and backup all 6 right now. Call them what you like and put them where you like, but make them. All 6 of FCC’s configuration files are in XML format. They’re not difficult to edit, but it is surprisingly easy to make a small mistake that will prevent the related program from starting. Having a backup will give you a fallback to get you back to a working setup.

XML files are just text files so you can use your favorite editor. We suggest one that supports colorization so it will be easier to visualize, but you can use NotePad (NotePad++ is better) if you’d like. If you decide use Word or its ilk be very careful that you don’t save the file in any format other than plain ‘ole text. Otherwise the file will become undecipherable to the service that needs it to run. If you find you can’t start a service after changing its configuration file you’d probably made a small mistake.

A note on creating comments in an XML file. I find it very handy to add comments to configuration files. Whenever I write a mod that reads a configuration file I allow a way to add comments to the file. You’ll find that FreedomPay has the same philosophy. Most of the parameters are preceded by a line or two of comments which describe the purpose of the setting and its valid values. I add comments to every configuration file setting I change. I lead with a header line on every change. Just look for “Infinity” to identify every setting I change. I comment out the original setting then add my new setting. That way we can identify what I changed, the original value and the new value.

To create a comment line, start the line with a less-than symbol, an exclamation point followed by 2 hyphens. That turns on commenting – to the end of the file – or until an end of comment sentinel is encountered. This is one reason I suggested using a more advanced editor that supports syntax coloring. It will show you which lines have been commented out. To turn commenting off type 2 hyphens followed by a greater-than sign. For example:

```
<!-- ***** This is a comment line ***** -->
```

You can create blocks of comments by entering the starting comment sentinel typing multiple lines of comments followed by the ending comment sentinel. But that's rather risky. You could easily delete, shuffle, copy/paste comment lines later and end up commenting out large portions of parameters. It's better to make each line its own separate comment line by starting each line with the start sentinel and ending with the end sentinel. Another reason to use an editor with syntax coloring.

But there's a drawback to adding additional comments to your configuration files. FreedomPay releases new versions of their software periodically and every version comes with its own new default configuration files. Most often the files are the same as the prior version. But sometimes they change a default parameter value or add a new parameter. It's up to you to ensure that your configuration files match up inasmuch as they have the same parameters albeit with different values. (Does that make sense ?) Adding comments to your configuration files makes it harder for you to perform the comparison. Initially I loaded my configuration files with comments because things were changing a lot and I had several environments that I needed to be able to configure for at any time. It was convenient to uncomment a line and comment another to make the changes. But each time FreedomPay released a new version I'd have to jump through hoops to compare my configuration files to their new default files. Eventually I gave it up. Now I merely load all of my comments in a single place in the file or keep a commentary version of the configuration file separate from the production version.

### **FCC Server Service (Figaro)**

After the installation Figaro will be installed and running – probably with the wrong configuration settings. So you might as well stop the service now and make a backup of the original configuration file if you didn't already. You can stop the service via the Services snap-in by clicking the Windows key (or clicking the Windows symbol in the lower-left corner of your screen) then typing "Services <ENTER>" in the "Search programs and files" text box. Or you can plod through Window's menus to get to it. On Windows 7 that's Start>Control Panel>System and Security>Administrative Tools>Services. Select "Freeway Server Service" then click "[Stop](#) the service".

You can also use the "net" command to stop the service from the command line. That looks like this:

```
net stop "FCCServerSvc"
```

If you look in the Datafiles directory of the CP-EMV installation (for Windows) you'll find three batch file that stop, start and restart the Susanna service. If you want you could use them as the basis for similar batch files for Figaro.

Figaro's configuration file is pretty sizable, but you only need to address a few settings if any. On my DropBox I have sample configuration files for several versions of the FCC. They all have ".sample" tacked on their filenames. Copy that file to the production configuration file's name FreewayServerService.exe.config. (Just drop the ".sample" at the end of the name.) You shouldn't need to make any changes to the sample Figaro configuration file I provide. (That's not true for the sample Susanna configuration file as we'll discuss later.)

Before you edit any of FCC's configuration files we should probably address a small permissions problem. Because you ran the FCC installation as an administrator any files and directories it created during the installation were created with the administrator as the owner. (Actually I'm not sure the files aren't delivered with those permissions.) That means you won't initially be able to save any changes you make to the configuration files. FreedomPay will probably address this in later versions, but for now that means you'll need to handle it yourself.

We've found 4 workarounds.

- Copy the configuration file to your desktop. This creates a new copy of the file with you as the owner. Then move/copy the file to the desired directory and make your changes.
- In Windows Explorer® right-clicked the file, select "Permissions, select the Security tab, click "Edit" to change the file's permissions, click "Users" in the top pane then check the "Full control" checkbox in the lower pane. Click "OK" then "OK" again.
- In Windows Explorer® right-clicked the file and select "Permissions". At the bottom of the "General" tab you should see an "Unblock" button. Click that followed by "OK".
- Open NotePad® (or another editor) as an administrator and you'll have the authority to edit/save the file without changing its permissions.

### *Figaro's Configuration File*

Open the file for editing. Our sample configuration file might be set up correctly for your immediate needs, but we'll touch on the basic settings you might need to understand. In the sample below we've stripped out the comment lines to compress the file into its relevant lines and omitted the second half of the file which you shouldn't modify at all.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="assetManagementSettings" allowExeDefinition="MachineToApplication"
type="FreedomPay.AssetManagement.Client.AssetManagementConfigurationSection,FreedomPay.AssetManagement.Client"
/>
  </configSections>
  <assetManagementSettings configSource="ama.config" />
  <appSettings>
    <add key="xmlPort" value="1011" />
    <add key="nvpPort" value="1012" />
    <add key="sslXmlPort" value="1013" />
    <add key="opiPort" value="1014" />
    <add key="positouchPort" value="3340" />
    <add key="httpProxyPort" value="5080" />
    <add key="httpProxyTimeout" value="300" />
    <add key="SSLPort" value="" />
    <add key="SSLThumbprint" value="" />
    <add key="islPort" value="3390" />
    <add key="clientPort" value="3391" />
    <add key="TLSSusannaConnection.clientPort" value="3392" />
    <add key="operationMode" value="serverAndWorkstation" />
    <add key="freewayServer" value="https://cs.freedompay.us/" />
    <add key="returnAllEmvTags" value="true" />
    <add key="freewayTimeout" value="35" />
    <add key="freewaySecondaryTimeout" value="60" />
    <add key="cardStorUrl" value="https://cs.freedompay.us/CardStor/CardStorService.asmx" />
    <add key="binRangeFile" value="C:\ProgramData\FreedomPay\Freeway Commerce Connect\HFEX.DAT" />
    <add key="binRangeURL" value="https://manager.freedompay.us/DCC/HFEX.txt" />
  </appSettings>
</configuration>
```

```

<add key="binRangeFileAge" value="1" />
<add key="binMapFile" value="C:\ProgramData\FreedomPay\Freeway Commerce Connect\BinMap.DAT" />
<add key="binMapURL" value="https://manager.freedompay.us/Map/BinMap.txt" />
<add key="binMapFileAge" value="1" />
<add key="eventLogName" value="FCCServerSvc" />
<add key="logFileName" value="C:\ProgramData\FreedomPay\Freeway Commerce Connect\log\FCC-server" />
<add key="logArchivalInterval" value="7" />
<add key="logDeleteInterval" value="30" />
<add key="showConsole" value="true" />
<add key="ClientSettingsProvider.ServiceUri" value="" />
<add key="clientTimeout" value="145" />
<add key="errorRepeatInterval" value="60" />
<add key="floorLimit" value="50" />
<add key="VerbalAuthFloorLimit" value="" />
<add key="maxLoggingQueueDepth" value="1000" />
<add key="saf.DeclineReplayForceKeyedCount" value="2" />
<add key="saf.DeclineReplayForceKeyed" value="true" />
<add key="offlineMaxDays" value="20" />
<add key="offlineCheckFrequency" value="5" />
<add key="offlineDeclineTime" value="02:30" />
<add key="offlineDeclinePurgeAge" value="10" />
<add key="databasePurgeAge" value="30" />
<add key="offlineFailureEnabled" value="true" />
<add key="offlineFailurePurgeAge" value="30" />
<add key="microsAcceptAllBatchRequests" value="false" />
<add key="logMaxFileLength" value="500000" />
<add key="receiptFormatter" value="legacy" />
<add key="printSignatureLine" value="true" />
<add key="signatureDisclaimer" value="" />
<add key="merchantCurrencyCode" value="USD" />
<add key="generateDumpFileOnCrash" value="true" />
<add key="crashDumpFileDir" value="C:\ProgramData\FreedomPay\Freeway Commerce Connect\dmp" />
<add key="FCC.MaxPacketSize" value="16384" />
<add key="enableCardNotPresentReceiptText" value="false" />
<add key="alwaysReturnIssuerName" value="true" />
<add key="merchantNameShort" value="Demo Res" />
<add key="merchantName" value="MICROS Demo Restaurant" />
<add key="address1" value="888 Test Ave., Columbia,MD" />
<add key="address2" value="" />
<add key="footer" value="I agree to the terms of my credit agreement." />
<add key="lineTerminator" value="#" />
<add key="pageTerminator" value="@" />
<add key="pageWidth" value="40" />
<add key="pageMargin" value="0" />
<add key="opiDccEnabled" value="true" />
<add key="opiTokenType" value="3" />
<add key="OPISupportFakeDetokenization" value="false" />
<add key="OpiZeroRoomRateOverride" value="100" />
<add key="Opi.LookupTokenByFolioNumber" value="false" />
<add key="VSCA.PrimaryPort" value="5015" />
<add key="VSCA.StoreID" value="" />
<add key="VSCA.TerminalID" value="" />
<add key="VSCA.RegisterNumber" value="" />
<add key="VSCA.QuickServeFloorLimit" value="100" />
<add key="VSCA.AllowPartial" value="true" />
<add key="VSCA.DPOI.Wait.Timeout" value="60000" />
<add key="VSCA.DPOI.ManualEntry.Wait.Timeout" value="60000" />
<add key="VSCA.DPOI.URL" value="http://127.0.0.1:8090/poi/" />
<add key="VSCA.Display.Format" value="7000" />
<add key="VSCA.ErrorMap" value="" />
<add key="VSCA.CheckDuplicatePayment" value="true" />
<add key="listenAddress" value="0.0.0.0" />
<add key="incrementalAuthType" value="reauth" />
<add key="useIpAsWorkstationId" value="false" />
<add key="EnableMpcLookup" value="false" />
<add key="settlementAddress" value="" />
<add key="returnUnknownIssuer" value="false" />
<add key="TLSSusannaConnection.sslThumbprint" value="" />
<add key="sslCertificateThumbprint" value="" />
<add key="httpStreamProxy.sslThumbprint" value="" />
<add key="httpStreamProxy.httpsThumbprint" value="" />
</appSettings>
</configuration>

```

<b>“nvpPort”</b>	This is the port that Figaro listens to for name/value pair formatted authorization requests. This is the port that CP-EMV uses so this port number is important. Leave it at 1012. You’ll find CP-EMV’s reference to this port in the file CP-EMV.INI.
<b>“clientPort”</b>	This is the port that Figaro listens for incoming Susanna connection requests. When a Susanna/Ingenico is powered up the corresponding Susanna calls out to Figaro to “introduce” itself and establish a connection. Leave it at 3391.
<b>“operationMode”</b>	Controls the way Figaro handles incoming authorization requests. For our purposes this value needs to be “serverAndWorksation”. The default Figaro configuration file from FreedomPay has this field set to “standalone”. That won’t work.
<b>“freewayServer”</b>	This is a very important setting. It controls which gateway Figaro uses to authorize voids and offline authorization requests. There are two valid values for this field. One directs authorization requests to their testing gateway. The other sends requests to their production gateway. We and FreedomPay both default our configuration files to the production gateway. If you want to perform testing against FreedomPay’s test gateway you’ll need to change this parameter. But in order to work with the testing gateway you’ll need to have Ingenicos that are keyed to use the testing gateway. Unless you specifically ordered Ingenicos keyed for a test environment you will have been sent production-keyed Ingenicos. So it’s probably safe to say you didn’t and so you won’t change this parameter. But to be complete, here are the only two valid values for this parameter.  Production gateway: <a href="https://cs.freedompay.us/">https://cs.freedompay.us/</a> Testing gateway: <a href="https://cs.uat.freedompay.com/">https://cs.uat.freedompay.com/</a>
<b>“freewayTimeout”</b>	This parameter sets how long Figaro will wait for a response from the gateway. The default value for this setting is 35 seconds which should be plenty. This value only applies to voids, offline requests (which are authorized by Figaro after the fact) and requests that don’t involve an Ingenico. That last example means authorization requests in which a token is provided in the request so a physical card need not be present. These types of transactions are all handled by Figaro and are not passed on to Susanna.
<b>“cardStorUrl”</b>	This parameter is paired with the freewayServer parameter. It determines which Freeway database to use to store and retrieve tokens from. If the freewayServer parameter is pointing to the production gateway, this parameter should be pointing to the production token database. Like the freewayServer parameter it has two valid values. Also like the freewayServer parameter you should leave it pointing at the production value.  Production: <a href="https://cs.freedompay.us/CardStor/CardStorService.asmx">https://cs.freedompay.us/CardStor/CardStorService.asmx</a> Testing: <a href="https://cs.uat.freedompay.com/CardStor/CardStorService.asmx">https://cs.uat.freedompay.com/CardStor/CardStorService.asmx</a>

**“logFileName”** This parameter provides the path and the first part of Figaro’s log file name. Figaro creates a new log file at the start of the first transaction on a new date. Note that the filename ends with “.../FCC-server”. Figaro adds to this a hyphen followed by the system date formatted “yyyy-mm-dd.xml”. Figaro then adds an underscore character followed by a sequence number starting with “1”. So this parameter value results in Figaro log files with names like “FCC-server-2020-04-26\_1.xml”. If the file grows in size to exceed the logMaxFileLength parameter value (discussed next), the file is closed and another log file with a similar name is created. This second file has the same name, but the sequence number is incremented. Note that the log file’s extension is “.xml”. Figaro’s log files are text files, but they don’t read like one. They are formatted in XML which makes them a little harder to read, but they still have information you need to perform debugging.

**“logArchivalInterval”** This parameter determines how long a log file is kept in text form before it is automatically zipped to save disk space. The default is seven days which is fine.

**“logDeleteInterval”** This parameter determines how long the log file is retained before being deleted automatically. The default is 30 days which might be a bit short. Sometimes a merchant doesn’t know something happened that requires researching the log file until they receive a statement. By then the 30 days might have passed and the associated log files have been deleted. You might want to bump this parameter up a bit, but we left it as is.

**“logMaxFileLength”** This parameter restricts the size of Figaro’s log file. Depending on the volume of traffic Figaro’s log file may grow in size to be difficult to open using typical editors. The default segmentation size is one million bytes. That’s a bit large for some of the editors my clients have installed (typically NotePad) so we’ve changed the parameter to half a meg like this.

```
<add key="logMaxFileLength" value="500000"/>
```

The value “500000” directs Figaro to limit its log files to 500,000 bytes. That will make the files smaller and easier (or possible) to open, but it may lead to more log file to open when investigating problems.

**“floorLimit”** Controls whether offline authorizations is active and what the threshold amount is. If this parameter is “0” offline authorizations is turned off. Any communication errors experienced by Susanna will result in Figaro declining the authorization. If a non-zero value is supplied (it must be positive integer) Figaro will generate a “fake” authorization when Susanna reports a communication error to the gateway if the transaction amount is not greater than the parameter value. The default configuration file has a “0” in this parameter, turning offline authorizations off.

Having said all that I need to say that this parameter is overridden by a similar value passed by CounterPoint to Figaro as part of the authorization request. See the paragraphs titled “Setup>Point of Sale>Stores>Configuration options” in the section “Modifications to CounterPoint” below.

- “logArchivalInterval”** Log files can become large and numerous. To reduce the impact on your hard drive FCC automatically zips Figaro log files after a user-defined number of days. This setting is that number of days.
- “logDeleteInterval”** In addition to zipping log files, FCC will also delete log files (even after they’re zipped) after a user-defined number of days. This setting is that number of days.
- “clientTimeout”** This setting is very important in the timing of an authorization. This parameter sets the number of seconds Figaro will wait for an authorization’s results to come back once it’s been sent to Susanna. There are several considerations to take into account when setting this field. The customer must swipe their card, the authorization must take place and the customer might have to sign the Ingenico. We’ve defaulted this field to 145 seconds. 60 seconds for the swipe, 20 seconds for the authorization, 60 seconds for the signature and 5 seconds for transmitting the request to and from Susanna. To this setting we added another 5 seconds to arrive at 150 seconds for CounterPoint’s timeout setting. If you change a timing setting here, in Susanna or in CounterPoint you need to recalculate what the new timeout needs to be in all three areas and change them all as appropriate. You cannot change just one or two. All three timings work in unison and must be set up with sensible values or transactions can be lost.

Once you’ve made the desired changes restart Figaro by clicking “[Start](#) service” in the Services Snap-in. (You didn’t already close the Services snap-in window did you ?)

Periodically FreedomPay will release a new version of FCC. Procedurally, installing the new version is exactly the same as when you installed the initial version. But as mentioned earlier you’ll want to download the Upgrade version of the installation if it’s available – not the Full version. Plus you’ll want to copy your existing Figaro and Susanna configuration files to the DISK1 directory prior to running the install. Don’t forget to compare your configuration file to the new default file to spot new, changed or deleted keys in the new file. You’ll need to incorporate new keys and evaluate whether you need to change existing keys that have changed. Just run the new executable as an administrator and the installation program will identify the components already installed and update them.

## FCC Client Service (Susanna)

After the installation Susanna will be installed and running – with the wrong configuration settings. For certain. So you might as well stop the service now and make a backup of the original configuration file if you didn't already. You can stop the service via the Services snap-in by clicking the Windows button then typing "Services <ENTER>". Or you can plod through Window's menus to get to it. Select "Freeway Client Service" then click "Stop the service".

You can also use the "net" command to stop the service from the command line. That looks like this:

```
net stop "FCCClientSvc"
```

If you look in the Datafiles directory of the CP-EMV installation (for Windows) you'll find three batch file that stop, start and restart the Susanna service. You can use them from the command line or create desktop icons to run them for convenience.

### *Susanna's configuration file*

Susanna's configuration file is even larger than Figaro's and has more parameters to set up. We provide a sample Susanna configuration file named FreewayClientService.exe.config.sample on our DropBox for each version of the FCC. Copy that file to the production configuration file's name FreewayClientService.exe.config. (Just remove ".sample" at the end of the name.)

As mentioned before (in the Figaro configuration file section) there's a small permissions problem associated with the FreedomPay configuration files security settings. Because you ran the FCC installation as an administrator any files and directories it created during the installation were created with the administrator as the owner. That means you won't initially be able to save any changes you make to the configuration files.

We've found 4 workarounds.

- Copy the configuration file to your desktop. This creates a new copy of the file with you as the owner. Then move/copy the file to the desired directory and make your changes.
- In Windows Explorer® right-clicked the file, select "Permissions, select the Security tab, click "Edit" to change the file's permissions, click "Users" in the top pane then check the "Full control" checkbox in the lower pane. Click "OK" then "OK" again.
- In Windows Explorer® right-clicked the file and select "Permissions". At the bottom of the "General" tab you should see an "Unblock" button. Click that followed by "OK".
- Open NotePad® (or another editor) as an administrator and you'll have the authority to edit/save the file without changing its permissions.

Open the file for editing. Most of the parameters in the sample configuration file are probably correct for your needs, but we'll touch on the basic settings you might need to understand. You'll definitely need to change at least one – the workstationIdMethodData parameter.

In the sample below we've stripped out the comment lines to compress the file into its relevant lines. We also shortened some of the lines to fit on a single line in the document. We've highlighted lines that you might have an interest in.



```

<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="assetManagementSettings" allowExeDefinition="MachineToApplication"
type="FreedomPay.AssetManagement.Client.AssetManagementConfigurationSection,FreedomPay.AssetManag
ement.Client" />
  </configSections>
  <assetManagementSettings configSource="ama.config" />
  <appSettings>
    <add key="logLevel" value="Debug4" />
    <add key="freewayUrl" value="https://cs.freedompay.us/Freeway/Service.asmx" />
    <add key="cardStorUrl" value="https://cs.freedompay.us/CardStor/CardStorService.asmx" />
    <add key="freewayTimeout" value="20" />
    <add key="posRequestServerUrl" value="http://127.0.0.1:1011" />
    <add key="posRequestServerAlterUrl" value="" />
    <add key="storeId" value="XXXXXXXXXX" />
    <add key="terminalId" value="XXXXXXXXXX" />
    <add key="simpleTerminalOverride" value="" />
    <add key="uriBasedConfig" value="true" />
    <add key="configURI" value="servers.xml" />
    <add key="binRangeFile" value="%PROGRAMDATA%\FreedomPay\Freeway Commerce Connect\HFEX.DAT" />
    <add key="binRangeURL" value="https://manager.freedompay.us/DCC/HFEX.txt" />
    <add key="binRangeFileAge" value="1" />
    <add key="binMapFile" value="%PROGRAMDATA%\FreedomPay\Freeway Commerce Connect\BinMap.DAT" />
    <add key="binMapURL" value="https://manager.freedompay.us/Map/BinMap2.txt" />
    <add key="binMapFileAge" value="1" />
    <add key="laneTimeout" value="60" />
    <add key="workstationIdMethod" value="ConfigFile" />
    <add key="workstationIdMethodData" value="1001" />
    <add key="forceCredit" value="false" />
    <add key="allowPartial" value="true" />
    <add key="teliumDefaultDeviceType" value="rba" />
    <add key="tetraDefaultDeviceType" value="upp" />
    <add key="traceInputEvents" value="false" />
    <add key="useMachineName" value="false" />
    <add key="useIPDevices" value="false" />
    <add key="useBluetoothDevices" value="false" />
    <add key="clientListens" value="false" />
    <add key="createTokenFromConfig" value="false" />
    <add key="tokenType" value="2" />
    <add key="signatureFormat" value="png" />
    <add key="signaturePrompt" value="Please Sign Here" />
    <add key="quickServeFloorLimit" value="50" />
    <add key="logDir" value="%PROGRAMDATA%\FreedomPay\Freeway Commerce Connect\log" />
    <add key="compressLogsAfter" value="7" />
    <add key="deleteLogsAfter" value="30" />
    <add key="logCompressMode" value="zip" />
    <add key="deviceProfileLocation" value="%PROGRAMDATA%\Freed ... onnect\pal\profile.dat" />
    <add key="palCheckInterval" value="0" />
    <add key="offHoursStart" value="11pm" />
    <add key="offHoursEnd" value="4am" />
    <add key="palVerboseLogging" value="true" />
    <add key="dccMCDDisclaimer" value="Please debit my ... tional transaction selection fee." />
    <add key="dccVSDDisclaimer" value="I have been offered ... is provided by the merchant." />
    <add key="dccPreAuthDisclaimer" value="" />
    <add key="dccDefaultDisclaimer" value="" />
    <add key="dcc.LocalCurrency" value="USD" />
    <add key="dcc.LocalCurrencyCode" value="840" />
    <add key="dcc.Paragraph1" value="" />
    <add key="dcc.Paragraph2" value="" />
    <add key="dcc.PromptLogic" value="BinInclude" />
    <add key="Offline8A00" value="true" />
    <add key="promptForTip" value="false" />
    <add key="tipPrompts" value="15,18,20" />
    <add key="generateDumpFileOnCrash" value="true" />
    <add key="crashDumpFileDir" value="%PROGRAMDATA%\FreedomPay\Freeway Commerce Connect\dmp" />
    <add key="DisableCreateTokenRequests" value="false" />
    <add key="poi.DisplayRoomRate" value="true" />
    <add key="FCC.MaxPacketSize" value="16384" />
    <!-- ***** -->
    <!-- DO NOT MODIFY ANYTHING BELOW THIS LINE -->
    <add key="SusannaAgentBase" value="net.tcp://127.0.0.1:8089/SusannaAgent/" />
    <add key="listenAddress" value="0.0.0.0" />
    <add key="listenPort" value="8090" />
    <add key="listenSSLThumbprint" value="739D6C35B9E8300F1248627FB2AD6F494B20A756" />
  </appSettings>

```

```

<add key="multiLaneMode" value="false" />
<add key="clientId" value="" />
<add key="concurrentLanes" value="false" />
<add key="enableHostedWCF" value="true" />
<add key="returnAllEmvTags" value="true" />
<add key="useIpAsWorkstationId" value="false" />
<add key="EnableMpcLookup" value="false" />
<add key="monikerPrefix" value="" />
<add key="forwardApprovalDate" value="true" />
<!-- DO NOT MODIFY ANYTHING ABOVE THIS LINE -->
</appSettings>
<!-- DO NOT MODIFY ANYTHING BELOW THIS LINE -->
<system.serviceModel>
  <bindings>
    <netTcpBinding>
      <binding name="SusannaAgentBinding" sendTimeout="00:01:00">
        <security mode="None" />
      </binding>
    </netTcpBinding>
  </bindings>
</system.serviceModel>
<!-- DO NOT MODIFY ANYTHING ABOVE THIS LINE -->
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="Freeway.Clients" publicKeyToken="9c59aafcf2fcef99"
culture="neutral" />
      <bindingRedirect oldVersion="0.0.0.0-4.8.2020.607" newVersion="4.8.2020.607" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
<startup>
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
</startup>
</configuration>

```

Below is an amplification of most of the parameters in this file.

**“freewayUrl”** This is a very important setting. There are two valid values for this field. One directs authorization requests to their testing gateway. The other sends requests to their production gateway. We and FreedomPay both default our configuration files to the production gateway. If you want to perform testing against FreedomPay’s test gateway you’ll need to change this parameter. But in order to work with the testing gateway you’ll need to have Ingenicos that are keyed to use the testing gateway. Unless you specifically ordered Ingenicos keyed for a test environment you will have been sent production-keyed Ingenicos. So it’s probably safe to say you didn’t and so you won’t change this parameter. But to be complete, here are the only two valid values for this parameter.

Production:      <https://cs.freedompay.us/Freeway/Service.asmx>  
Testing:      <https://cs.uat.freedompay.com/Freeway/Service.asmx>

**“freewayTimeout”** This is the number of seconds Susanna will wait for a response from the gateway. FreedomPay’s default value is 35 seconds, but typically the actual response time is less than a couple of seconds. If Susanna doesn’t get a reply in 20 seconds something has gone wrong and it’s not going to get a response. Why wait even longer for the inevitable error ? So we changed the parameter to 20 seconds which should be plenty of time. Leave this parameter as-is unless you are having problems. If you change it to a higher value you’ll need to change Figaro’s “clientTimeout” setting and the Timeout value in each store’s configuration settings.

**“cardStorUrl”** This parameter is paired with the freewayUrl parameter. It determines which Freeway database to use to store and retrieve tokens from. If the freewayUrl parameter is pointing to the production gateway, this parameter should be pointing to the production token database. Like the freewayUrl parameter it has two valid values. Also like the freewayUrl parameter you should leave it pointing at the production value.

Production: <https://cs.freedompay.us/CardStor/CardStorService.asmx>

Testing: <https://cs.uat.freedompay.com/CardStor/CardStorService.asmx>

**“storeId”** You might recognize this parameter and know it needs to have a value assigned. And it does, but not here. This parameter is stored in CounterPoint’s store configuration record and passed along with the authorization request. Don’t put a value here. Leave this parameter with FreedomPay’s default setting.

**“terminalId”** You might recognize this parameter and know it needs to have a value assigned. And it does, but not here. This parameter is stored in CounterPoint’s store configuration record and passed along with the authorization request. Don’t put a value here. Leave this parameter with FreedomPay’s default setting.

**“laneTimeout”** This is the number of seconds Susanna will wait for a response from the Ingenico. That means the number of seconds the customer has to swipe their card and separately the number of seconds to provide a signature. FreedomPay defaults this field 90 seconds. When calculating total authorization timings you need to consider this timing happens twice. Once to swipe the card and possibly again to provide a signature. So 90 seconds times 2 is 3 minutes. That’s a long time and not even the total round trip time. We changed this parameter to 60 seconds which should be plenty of time for grandma to swipe her card or sign the signature pad. Leave this parameter as-is unless you are having problems. If you change it to a higher value you’ll need to change Figaro’s “clientTimeout” setting and the Timeout value in CP-EMV.INI.

- “workstationIdMethod”** This parameter determines the method used to define the workstation’s ID to Figaro. It can contain one of several possible values, but we’re only interested in one – “ConfigFile”. Use that value.
- “workstationIdMethodData”** This parameter is what Figaro uses to map an incoming authorization request to a particular register. It is a 6 digit number. The first 3 digits represent the CounterPoint store number. The last 3 digits are the register number. So the value of “146002” represents register “2” at store number “146”. Drop leading zeros so store “1” register “1” is entered as “1001” (the sample value). If you get this wrong you’ll get a “Workstation not found” error or worse, authorization requests for that register will go to another register – maybe a register at a different store !
- “forceCredit”** This parameter tells Susanna to force debit cards to be processed as credit cards. This is more convenient because a PIN isn’t prompted for, but the transaction rate might be slightly higher. Note that you can’t just change this parameter to enact the change to PIN processing. When you boarded with FreedomPay there was a checkbox on the form as to whether you wanted to support PIN-enabled devices. That determined how the Ingenico was setup. If you didn’t check that box then merely setting this to “false” won’t enable PINs. The default is “false” forcing the Ingenico to make the decision.
- “allowPartial”** This “true”/“false” parameter controls whether the workstation allows partial payments. We thought supporting partial payments was required by the major processors so we coded CP-EMV to support this. So we left the value for this parameter to the default value of “true”.
- “teliumDefaultDeviceType”** Tells Susanna the type of interface to use with the connected Ingenico.
- “tetraDefaultDeviceType”** Tells Susanna the type of interface to use with the connected Ingenico.
- “traceInputEvents”** Controls detailed logging of the transaction process. This is used by FreedomPay and only rarely. Leave it set to “false” unless instructed otherwise.

- “useMachineName”** These two parameters work as a an either/or pair. In prior versions of Susanna specifying one of these two parameters controlled whether the workstationId was hardcoded or whether the machine’s network name is used. I would think the new parameters “workstationIdMethod” and “workstationIdMethodData” would replace these parameters, but here they are still included. The parameter “useMachineName” is an alternative to using the workstationId parameter, but that won’t work for us. Figaro needs to be able to uniquely identify each register and CounterPoint needs to be able to generate and pass to Figaro that identifier. FreedomPay’s default Susanna configuration file specifies “useMachineName” is “true”. So we’ve disabled that line in our sample configuration file. In our testing we disabled the “workstationID” parameter and restarted Susanna. It seemed to use the value we specified for the “workstationIdMethodData” parameter as expected. So we’ve disabled both of these parameters in our sample files.
- “useIPDevices”** This parameter is used when your Ingenicos are attached via network cables to the network switch (as opposed to being attached to the register via a USB cable). FreedomPay’s (and our) default value is “false” since almost every uses USB cabled Ingenicos. But we do have a single client with many stores all using dumb terminals with networked Ingenicos so we have to discuss this parameter here.
- “useBluetoothDevices”** This “true”/“false” parameter is similar to the “useIPDevices” parameter. It tells Susanna to communicate with the Ingenico via Bluetooth instead of USB or a network connection. FreedomPay’s default value is “false” and we agree.
- “clientListens”** This parameter allows Susanna to listen for authorization requests directly. Leave this set to the default “false”.
- “createTokenFromConfig”** This parameter forces the gateway to generate tokens for every authorization request. FreedomPay’s default value is “false”. Previously this parameter didn’t exist and all authorization requests returned a token. CP-EMV would copy this token to its history files to facilitate cardless returns. Because this configuration file parameter can be overridden by an equivalent parameter in the authorization request we suggest leaving this parameter set to “false”.
- “tokenType”** This parameter is used to determine what format tokens should be stored. We pass this value as part of the authorization request which should override this setting, but considering how important it is we figured we should include a brief discussion of it here. Its value should be “2”.

- “signatureFormat”** This “secret” parameter is used to determine what format signatures should be returned in. When a customer provides a signature the request response back to CounterPoint includes a coded text string that needs to be reformatted to create an image of the signature. This setting tells the Ingenico how to code that image. If you change this parameter, the signature files will contain garbage. The default client configuration file doesn’t have this parameter in it, but the underlying code in FreedomPay is still in there and working so we’re using it to our advantage. We assign it the value “png”.
- “signaturePrompt”** This parameter provides the prompt for “special” signature operations. I’m not exactly sure what “special” means so leave this parameter as is.
- “quickServeFloorLimit”** Merchants often decide not to require their customers to have to sign for an authorization if the transaction amount falls below a certain threshold. This is the parameter that controls that threshold. If you set this parameter to “0” you are requiring customers to always sign (if the Ingenico is signature capable). The default value is “50” meaning all transactions less than \$50 will not be prompted for a signature.
- “logDir”** Points to the directory in which Susanna’s log files are written. Unlike Figaro this parameter is only the path, not the path and the first part of the log file’s name..
- “compressLogsAfter”** This parameter determines how long a log file is kept in text form before it is automatically zipped to save disk space. The default is seven days which is fine.
- “deleteLogsAfter”** This parameter determines how long the log file is retained before being deleted automatically. The default is 30 days which might be a bit short. Sometimes a merchant doesn’t know something happened that requires researching the log file until they receive a statement. By then the 30 days might have passed and the associated log files have been deleted. You might want to bump this parameter up a bit, but we left it as is.
- “logCompressMode”** This parameter allows the client to specify the type of compressed file to be used. Unless you have a reason not to use .zip files leave this parameter set its default value of “zip”.

**“deviceProfileLocation”** Now that FreedomPay has developed their PAL update process, Susanna needs to know where to find the files associated with all of that. This parameter provides the top-level directory for that. Their default location is “C:\ProgramData\FreedomPay\Freeway Commerce Connect\pal” and we’re fine with that so we didn’t change it. But you might have some reason to. You could put a network location here that all of your registers have access to the same files. Thus you wouldn’t need to copy the files to each machine. Just put the update files in one place and all of the Ingenicos will get them from the one place. For the complete PAL documentation look for the FreedomPay document “POI Application Loader” on my DropBox in the directory “Ingenicos”.

**“palCheckInterval”** This parameter indicates the number of minutes between checking for PAL updates. You really don’t want to allow your Ingenicos to update on their own. We changed this parameter from the default “5” to “0” in order to shut automatic updates off.

**“offHoursStart”** This parameter sets the time to start trying to update the Ingenico.

**“offHoursStop”** This parameter sets the time to stop trying to update the Ingenico.

**“palVerboseLogging”** This parameter controls the quantity of messages PAL writes to the Susanna log file as it operates. The default is “true” which tells me FreedomPay Tech Support really does want this information. So we left it as is.

Beyond this point in the Susanna configuration file there are many more parameters that are either poorly documented or mean absolutely nothing to CP-EMV. However there are two parameters in this section

**“multiLaneMode”** This parameter allows Susanna to control multiple Ingenicos. This ability doesn’t make much sense unless your store employs “dumb” (non-Windows) terminals as registers. Unless you are operating in this environment leave this parameter set to “false”.

**“returnAllEmvTags”** At some point the FCC stopped returning all the fields CP-EMV parses for use on receipts. This parameter forces Susanna to provide those fields. The default value is “false”. We changed it to “true” in our sample files.

Whenever you make a change to this file you need to restart Susanna before the service reads the file from disk and the changes take effect.

## *Servers.xml*

This is an important file. In order to expedite rolling out groups of Susannas, FreedomPay wrote Susanna to retrieve a few of its more common settings from a separate file named Servers.xml. Susanna is configured to use this file and you should leave it that way. The settings in the file supersede the same settings found in Susanna's configuration file. The file looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<SusannaConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <FigaroServers>
    <FigaroServer server="127.0.0.1" port="3392" tls="true" tlsHostname="ED-8570P" />
  </FigaroServers>
  <Credentials>
    <Credential storeID="123456789" terminalID="1234567890" />
  </Credentials>
</SusannaConfiguration>
```

The servers.xml file that FreedomPay creates has all of the text on a single line. While this is completely acceptable to a computer it certainly makes it harder for a human to decipher the contents of the file. So we've added LF/CRs to the file to put each "line" on its own line. That version is what you see above and is what we provide in our sample files.

We've bolded and highlighted the one important line in this sample. The parameters "server" and "port" point to Figaro. This example is using the "loop back" address "127.0.0.1". In networking terms that means "this machine". You can put a local IP address (192.168.xxx.xxx) if Figaro is connected to the local network, a public IP address (like 42.232.124.223) if the machine is off site. You can even employ a URL such as "cc.myfigaro.com" and Susanna will perform a DNS lookup to get the actual IP address.

The parameter "tlsHostname" might prove to be a problem though. When installing Susanna the installer apparently poles the machine and loads its machine name into this parameter. That might mean we can no longer use the same servers.xml file for all registers in a store.

The parameters "storeid" and "terminalid" provide merchants credentials as assigned by FreedomPay. But since they are overridden by the same fields passed in the authorization request there's no reason to change the default values. In fact it would be a bad idea to put such sensitive information in such an easily accessible text file.

Like the Susanna configuration file, whenever you make a change to this file you must restart Susanna before the changes will take effect.



## Updating the Ingenico

Susanna is in charge of keeping its attached Ingenico's software up to date. FreedomPay has created a couple of automated processes to facilitate that and yet allow you considerable control over the process. The first was called "FileWrite" and is now considered obsolete. I've left the documentation for that below for completeness, because it still works and because it took me so much time to write. But you really should be using FreedomPay's newer "POI Application Loader" process named "PAL". A description of that process is below the documentation for FileWrite. For the official documentation on using FileWrite and PAL, look for the FreedomPay document "POI Application Loader" on my DropBox in the directory "Ingenicos".

### *FileWrite (versions 1.0 and 2.0)*

The parameters that control the FileWrite update process are in Susanna's configuration file. They are: "autoDeviceUpdate", "deviceUpdateTime", "rebootNeededStartTime", "rebootNeededStopTime" and "remoteUpdateFiles". Once you've set up the files and the parameters you can wait for the update to happen or you can force the update by restarting Susanna, rebooting the machine or via the FCC-UI.

The first, most important file in the update process is the manifest file named FileWrite.xml which resides in the "remoteUpdateFiles" directory. It drives the update process in several regards.

- It sequences the update process.
- It maps specific update files to the Ingenico model they serve.
- It controls rebooting the Ingenico after updating certain types of files.
- It allows controlling minimum/maximum version numbers to be updated.

Here is the sample FileWrite.xml that is delivered with FCC.

```
<Updates>
  <!-- version 1.0 -->
  <Update deviceName="rba-iSC250" minVersion="1511" maxVersion="1512" type="firmware" fileName="RGEN031512.OGZ"/>
  <Update deviceName="rba-iSC250" minVersion="1511" maxVersion="1512" type="rfile" fileName="UP012916C.TGZ"/>
  <Update deviceName="rba-iSC250" minVersion="1511" maxVersion="1512" type="ifile" fileName="AD2.JPG"/>
  <Update deviceName="rba-iSC250" minVersion="1511" maxVersion="1512" type="ifile" fileName="AD3.JPG"/>
</Updates>
```

"deviceName" Indicates the model of Ingenico. That includes the type of software running on the Ingenico. All Ingenicos running with CP-EMV use the "Retail Base Application" (RBA) thus all of the device names you'll use will start with "rba-". The sample refers to an iSC250.

"minVersion" This refers to the minimum RBA version number to be loaded on the Ingenico. It's not the version of FCC (or CounterPoint). The sample refers to "1511" which translates to version 15.11. (The decimal point is dropped.)

"maxVersion" This refers to the maximum RBA version number to be loaded on the Ingenico. The sample refers to "1512" which translates to version 15.12. (The current version as of now.)

"type" This classifies the update file. There are four possible values for this field.

- “`firmware`” A firmware file (\*.OGZ) that forces the Ingenico to be rebooted immediately after it has been installed.
- “`rfile`” A configuration file (\*.TGZ) that forces the Ingenico to be rebooted immediately after it has been installed.
- “`ifile`” Indicates a file that does not require the Ingenico to be rebooted after being installed. Typically this would be a JPG file for updating branding.
- “`file`” A file that is scheduled for a reboot after a prerequisite file is updated.

“`filename`” This is the name of the update file.

When the system time hits the value for the “`rebootNeededStartTime`” Susanna starts trying to update the Ingenico. It looks for `FileWrite.xml` in the directory pointed at by the “`remoteUpdateFiles`” parameter. If it finds one it begins processing the file from top to bottom. It already knows what type of Ingenico is attached to it so it’s looking for lines in `FileWrite.xml` that pertain to it. As it encounters each relevant line it processes the line as required. This might require the Susanna to reboot the Ingenico before looking for another line to process. It does this until it hits the end of the file.

If the update file being processed is a type “`firmware`” or “`rfile`” file (a .TGZ or .OGZ file) the update process copies it into the “`Reboot`” directory within FCC’s executable directory. An \*.OGZ file carries the Ingenico’s firmware and/or RBA. Like any other program, if a new version of a program is installed it must be restarted before the new program will take effect. So the Susanna commands the Ingenico to reboot. A \*.TGZ file provides the configuration settings for the installed RBA. For the same reason Susanna commands the Ingenico to reboot after it’s been installed.

If the update file is a type “`ifile`” (typically a \*.JPG file) the update process copies it into the “`NoReboot`” directory within FCC’s executable directory. A \*.JPG file is an image file that is displayed on the Ingenico during periods of inactivity. They are not critical to the operation of the Ingenico (they’re not required at all) and don’t require the Ingenico to be rebooted. You can specify to upload more than one advertising file. But they need to be named AD1.jpg through AD9.jpg. Other filenames will be ignored. Different models of Ingenicos have different screen resolutions and so have different optimum advertising image dimensions.

iPP320	128x64 (Black & white LCD screen doesn’t support JPG files)
iPP350	320x240 (Color)
iSC250	480x272 (Color)
iSC480	800x480 (Color)

With each step Susanna updates its log file so you have an audit trail that it happened and whether it was successful. The update file stays in the “`Reboot`” directory. Susanna has to keep track of that or it will try to install the update file again (and again).

Susanna attempts this update process every “`deviceUpdateTime`” minutes starting at “`rebootNeededStartTime`” and ending at “`rebootNeededEndTime`”.

The default value for autoDeviceUpdate is “false”. So automatic updates are turned off “out of the box”. But you really should update you Ingenicos to the latest software available at least at the start. You don’t need to turn automatic updates on to make this happen. You could use the FCC-UI’s “File Write” button to perform an immediate update. But you do need to set up Susanna’s configuration file for updates and download the latest files. If you do turn on automatic updates you may want to edit the file to change the start and end times or you can let them sit overnight and they’ll update between 1am and 6am. (That’s the default settings.)

The “remoteUpdateFiles” directory need not be on the Susanna machine. For the best results assign this field the path to a network drive that all of your Susannas can access. Edit FileWrite.xml to have entries relevant to all of the Ingenicos your store uses. This makes it possible to download the latest update files to a single networked location and have all of your Ingenicos updated automatically overnight.

**Important Notice:** You may think that you don’t need to bother setting up automatic updates because you’re not going to turn them on anyway. Only part of that is true. You don’t have to set up the start/end times or “deviceUpdateTime”, but you absolutely should set up FileWrite.xml and “remoteUpdateFiles”. It is quite probable that you’re going to update your Ingenicos eventually. You should at least update them when you first install them. Surely you’ll update any replacement units you receive later. Will you remember that you never set up properly to update your Ingenicos then ? If you update your Ingenicos with a malformed FileWrite.xml file you run the risk of “bricking” one or all of your Ingenicos. Once an Ingenico has been bricked it can only be repaired by shipping it back to ScanSource for reloading. Take the time to set up FileWrite.xml or make absolutely certain that it is missing from your “remoteUpdateFiles” directory.

If you need to force FCC-UI to perform an Ingenico update twice you may find that clicking the “Write Device” button won’t start the update anymore. There are a couple of reasons for this.

1. Susanna calculates a “hash” value of the FileWrite.xml file prior to performing the update. If the hash value for the current FileWrite.xml file is the same as the value of the last update run, Susanna won’t run the update. When this is the problem you’ll see the message “No manifest was found or no new manifest was found.” in Susanna’s log file. You need to change something in the new FileWrite.xml file to change its hash value. That will trick Susanna into thinking there’s a new manifest file to process.
2. Even when an update process is initiated from the FCC-UI Susanna still checks if the process is within the update start/end time. If one attempts to update an Ingenico outside of the defined start/end time the update won’t start and you’ll find “” in Susanna’s log file. You need to add the line “<ScheduleBypass>true</ScheduleBypass>” towards the top of the FileWrite.xml file like this.

```
<Updates>
  <ScheduleBypass>true</ScheduleBypass>
  <Update deviceName="rba-iSC250" minVersion="1510" maxVersion="1512" type="firmware" fileName="RGEN031512.0GZ"/>
  <Update deviceName="rba-iSC250" minVersion="1510" maxVersion="1512" type="rfile" fileName="UP041116C.TGZ"/>
  <Update deviceName="rba-iSC250" minVersion="1510" maxVersion="1512" type="ifile" fileName="AD2.JPG"/>
</Updates>
```

This will allow Susanna to perform updates outside the defined start end times.

We've placed various known-to-work versions of the RBA on our DropBox in the "Ingenicos" directory.

### ***POI Application Loader (PAL)***

This is the process you should use to update your Ingenicos. The parameters that control the automated update process are in Susanna's configuration file. They are: "deviceProfileLocation", "palCheckInterval", "offHoursStart", "offHoursEnd" and "palVerboseLogging". Once you've set up the files and the parameters you can wait for the update to happen or you can force the update by restarting Susanna or rebooting the machine.

The Susanna configuration parameter "deviceProfileLocation" points to the first file in the process. Since I recommend leaving this parameter as is I'm going to further assume that I can refer to all default file and directory names in this section.

In the directory C:\ProgramData\FreedomPay\Freeway Commerce Connect\pal you should find a file named profile.dat and two directories named "EMVContact" and "Firmware". Profile.dat drives the process and the two directories are where the updates happen. Profile.dat is a simple text file that has a line for each directory that should be inspected for an update. By default the file has two entries that match the names of the two directories - "EMVContact" and "Firmware". The directories are processed in the order they appear in the file. Typically, you'll only use the "Firmware" directory for performing updates. There's really no reason to change the profiles.dat file unless you add a new directory for updating things like the advertising files that display on your Ingenicos.

Initially each of the directories will have a single file – manifest.dat. This file is what controls the update process in that directory. It drives the update process in several regards.

- It sequences the update process.
- It maps specific update files to the Ingenico model they serve.
- It controls rebooting the Ingenico after updating certain types of files.
- It allows controlling minimum/maximum version numbers to be updated.

The manifest.dat files are different between the two directories. The EMVContact directory appears to be used to update a wireless model of an Ingenico (iWL250) which I know nothing about so I'm going to skip that directory and concentrate on the only PAL directory I've ever needed to touch – the Firmware directory. Here is the sample manifest.dat that is delivered with FCC version 4.2.2.6. Any line that starts with a hash tag is commentary. Almost all of the lines are commented out.

```
name=Firmware
description=Firmware images and configuration files
version=1.0.0.0

##### write an appropriate firmware image, OGZ #####

#Uncomment Following for iPP320 Devices to update to RBA 15.1.2
#firmware --hw=rba-iPP320 --minver=0000 --maxver=1510 RGEN041512.OGZ firmware.OGZ

#Uncomment Following for iPP350 Devices to update to RBA 15.1.2
#firmware --hw=rba-iPP350 --minver=0000 --maxver=1510 RGEN021512.OGZ firmware.OGZ

#Uncomment Following for iSC250 Devices to update to RBA 15.1.6
#firmware --hw=rba-iSC250 --minver=0000 --maxver=1512 R031516B3.OGZ firmware.OGZ
```

```

#Uncomment Following for iSC480 Devices to update to RBA 15.1.2
#firmware --hw=rba-iSC480 --minver=0000 --maxver=1510 RGEN311512.OGZ firmware.OGZ

#Uncomment Following for iMP352 Devices to update to RBA 15.1.2
#firmware --hw=rba-iMP352 --minver=0000 --maxver=1510 RGEN101512.OGZ firmware.OGZ

#Uncomment Following for iUP250_iUR250 Devices to update to RBA 15.1.2
#firmware --hw=rba-iUP250 --minver=0000 --maxver=1510 RGEN211512A.OGZ firmware.OGZ

#Uncomment Following for iUP250_iUR250_iUC250 Devices to update to RBA 15.1.2
#firmware --hw=rba-iUP250 --minver=0000 --maxver=1510 RGEN211512B.OGZ firmware.OGZ

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% write an appropriate configuration file, TGZ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

# UNCOMMENT ONLY ONE PER DEVICE TYPE

#***** USB MODE *****

#----- EMV ENABLED -----

#Uncomment Following to Write USB Mode, EMV ENABLED Data Package to iPP320s
#rfile --hw=rba-iPP320 --minver=1511 --maxver=1512 UP061716A.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV ENABLED Data Package to iPP350s
#rfile --hw=rba-iPP350 --minver=1511 --maxver=1512 UP061716B.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV ENABLED Data Package to iSC250s
#rfile --hw=rba-iSC250 --minver=1511 --maxver=1516 UP061716C.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV ENABLED Data Package to iSC480s
#rfile --hw=rba-iSC480 --minver=1511 --maxver=1512 UP061716E.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV ENABLED Data Package to iUP250_iUR250
#rfile --hw=rba-iUP250 --minver=1511 --maxver=1512 UP061716F.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV ENABLED Data Package to iUP250_iUR250_iUC250
#rfile --hw=rba-iUP250 --minver=1511 --maxver=1512 UP061716G.TGZ config.TGZ

#----- EMV DISABLED -----

#Uncomment Following to Write USB Mode, EMV DISABLED Data Package to iPP320s
#rfile --hw=rba-iPP320 --minver=1511 --maxver=1512 UP061816A.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV DISABLED Data Package to iPP350s
#rfile --hw=rba-iPP350 --minver=1511 --maxver=1512 UP061816B.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV DISABLED Data Package to iSC250s
#rfile --hw=rba-iSC250 --minver=1511 --maxver=1516 UP061816C.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV DISABLED Data Package to iSC480s
#rfile --hw=rba-iSC480 --minver=1511 --maxver=1512 UP061816E.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV DISABLED Data Package to iUP250_iUR250
#rfile --hw=rba-iUP250 --minver=1511 --maxver=1512 UP061816F.TGZ config.TGZ

#Uncomment Following to Write USB Mode, EMV DISABLED Data Package to iUP250_iUR250_iUC250
#rfile --hw=rba-iUP250 --minver=1511 --maxver=1512 UP061816G.TGZ config.TGZ

#***** ETHERNET MODE *****

#----- EMV ENABLED -----

#Uncomment Following to Write ETHERNET Mode, EMV ENABLED Data Package to iPP320s
#rfile --hw=rba-iPP320 --minver=1511 --maxver=1512 IP061716A.TGZ config.TGZ

#Uncomment Following to Write ETHERNET Mode, EMV ENABLED Data Package to iPP350s
#rfile --hw=rba-iPP350 --minver=1511 --maxver=1512 IP061716B.TGZ config.TGZ

#Uncomment Following to Write ETHERNET Mode, EMV ENABLED Data Package to iSC250s
#rfile --hw=rba-iSC250 --minver=1511 --maxver=1516 IP061716C.TGZ config.TGZ

```

```

#Uncomment Following to Write ETHERNET Mode, EMV ENABLED Data Package to iSC480s
#rfile --hw=rba-iSC480 --minver=1511 --maxver=1512 IP061716E.TGZ config.TGZ

#----- EMV DISABLED -----

#Uncomment Following to Write ETHERNET Mode, EMV DISABLED Data Package to iPP320s
#rfile --hw=rba-iPP320 --minver=1511 --maxver=1512 IP061816A.TGZ config.TGZ

#Uncomment Following to Write ETHERNET Mode, EMV DISABLED Data Package to iPP350s
#rfile --hw=rba-iPP350 --minver=1511 --maxver=1512 IP061816B.TGZ config.TGZ

#Uncomment Following to Write ETHERNET Mode, EMV DISABLED Data Package to iSC250s
#rfile --hw=rba-iSC250 --minver=1511 --maxver=1516 IP061816C.TGZ config.TGZ

#Uncomment Following to Write ETHERNET Mode, EMV DISABLED Data Package to iSC480s
#rfile --hw=rba-iSC480 --minver=1511 --maxver=1512 IP061816E.TGZ config.TGZ

#***** BLUETOOTH MODE *****

#----- EMV ENABLED -----

#Uncomment Following to Write BLUETOOTH Mode, EMV ENABLED Data Package to iMP352s
#rfile --hw=rba-iMP352 --minver=1511 --maxver=1512 BP061716D.TGZ config.TGZ

#----- EMV DISABLED -----

#Uncomment Following to Write BLUETOOTH Mode, EMV DISABLED Data Package to iMP352s
#rfile --hw=rba-iMP352 --minver=1511 --maxver=1512 BP061816D.TGZ config.TGZ

#***** SERIAL MODE *****

#----- EMV ENABLED -----

#Uncomment Following to Write SERIAL Mode, EMV ENABLED Data Package to iPP350s
#rfile --hw=rba-iPP350 --minver=1511 --maxver=1512 SP061716B.TGZ config.TGZ

#----- EMV DISABLED -----

#Uncomment Following to Write SERIAL Mode, EMV DISABLED Data Package to iPP350s
#rfile --hw=rba-iPP350 --minver=1511 --maxver=1512 SP061816B.TGZ config.TGZ

##### apply pending changes (i.e. if there's a reboot scheduled, do it now) #####
apply

```

The body of the file is divided in to 4 sections. The first three lines are the header. The next section is used to update the firmware called the Retail Base Application (RBA) of the Ingenico. This is the software that drives the basic processes that happen on the Ingenico. These types of update files end with the extension “.OGZ”. Those lines start with the word “firmware”. (Kinda obvious huh ?) The next section tailors to operation of the RBA through configuration settings. These lines start with “rfile” and have the extensions “.TGZ”. The last section is really just a single line that reads “apply”. This line commits the changes and restarts the Ingenico (if required).

Let's start with the header section.

"name" The name of the update. I assume it should be the same as the directory name. That's what I've seen so far.

"description" A commentary line of what's going on.

"version" This is important. Susanna builds a database (registry entries I believe) of the devices that have been updated by it not just by model number, but by serial number. As PAL updates an Ingenico, Susanna makes an entry that the Ingenico with serial number so-and-so has been updated to version such-and-such. That version number comes from this line. Each time PAL goes to update the attached Ingenico it compares this version number to the version number it has for the attached Ingenico. If the version number is the same as or lower than what has been stored for the Ingenico the update won't be performed. So each time you update this file you must increment this number. The default value is "1.0.0.0" so there's plenty of room to grow, but I wouldn't number the updates "1.0.0.0", "2.0.0.0", "3.0.0.0"... Give yourself more room than that.

The entries in the firmware and configuration sections have the same format. Let's take a look at a sample line.

```
firmware --hw=rba-iSC250 --minver=0000 --maxver=210A filename1 filename2
```

The first word determines the type of update the line is making. That's going to be "firmware" or "rfile". The parameters after that are pretty much the same.

"--hw=*model*" Indicates the model of Ingenico. That includes the type of software running on the Ingenico. All Ingenicos running with CP-EMV use the "Retail Base Application" (RBA) thus all of the device names you'll use will start with "rba-". So you'll use values like "rba-iPP320", "rba-iPP350", "rba-iSC250" and "rba-iSC480".

"--minver=*0000*" This refers to the minimum RBA version number that should already be loaded on the Ingenico. The idea is to ensure you don't attempt to load an update unless the Ingenico is already at some minimum version of the RBA. If the Ingenico's RBA is a version less than this value the update will not be performed. It's not the version of FCC (or CounterPoint). The default value "0000" means "Ignore this parameter". The first two characters are the main version number. Each character after that is a (more) minor sub-version number. Values over "9" are represented by capital letters. So RBA version 21.0.10 would be entered as "210A".

`--maxver=0000` This refers to the maximum RBA version number that should already be loaded on the Ingenico. The idea is to ensure you don't attempt to load an update unless the Ingenico is already at some lower version of the RBA. If the Ingenico's RBA is a version greater than this value the update will not be performed. It's not the version of FCC (or CounterPoint). The default value is whatever they last used in the file so pay attention. You need to ensure this parameter makes sense. The first two characters are the main version number. Each character after that is a (more) minor sub-version number. Values over "9" are represented by capital letters. So RBA version 21.0.10 would be entered as "210A".

`filename1` This is the name of the update file to be pushed into the Ingenico.

`filename2` The update process is coded to use the same name regardless of the name of the file being pushed. So the first thing PAL needs to do is copy the update file to the expected name. That name is different depending on whether a firmware or configuration file is being pushed. Firmware update files are named "firmware.OGZ" and so that is the value of `filename2`. Lines that push configuration files end with "config.TGZ".

To perform an update you need to update the manifest.dat file and copy the associated update file in the same directory. Open the manifest.dat file with your favorite editor. Increment the version field in the header. Find the line in the appropriate section of the file (firmware or configuration section) and remove the hash tag from the line for the device you're updating. Check that the minimum and maximum RBA version numbers are OK and change them if required. Then change the name of the update file (`filename1` in our example).

When the system time hits the value for the "offHoursStart" Susanna starts trying to update the Ingenico. It looks in the file indicated by the "deviceProfileLocation" parameter. If it finds one it begins processing the file from top to bottom. It reads each directory's manifest.dat file and determines if there's anything to do. As it encounters each relevant line it processes the line as required. This might require the Susanna to reboot the Ingenico before looking for another line to process. It does this until it hits the end of the file. It then waits for the next number of minutes as determined by "palCheckInterval" then repeats the process until "offHoursStop".

With each step Susanna updates its log file so you have an audit trail that it happened and whether it was successful. The Susanna configuration file setting "palVerboseLogging" controls how verbose these messages are.



FreedomPay defaults "palCheckInterval" to "5" meaning Susanna will check for updates every 5 minutes between "offHoursStart" and "offHoursStop". Our default value for palCheckInterval is "0". So automatic updates are turned off "out of the box". That's because we really don't want our client's Ingenicos updating themselves without someone doing it intentionally. But you really might want to update your Ingenicos now and then for security updates or newly added features. You don't need to turn automatic updates on to make this happen. You can set up the update (manifest.dat and update file changes) then force the update to happen by restarting the Susanna service or rebooting the machine (which is essentially the same thing). Whenever Susanna starts it runs PAL regardless of whether automatic updates are turned on. This can be convenient and evil. Don't leave a modified manifest.dat file and/or update files in your PAL directories thinking you'll get back them later. On the first restart those files are going to be processed before you can get in there and stop the process.

However automatic updates can allow you quite a convenience if you take the time to actually manage the process. Create a directory on a shared network drive pointed to by "deviceProfileLocation" on all of your registers. Edit the manifest.dat file and put the update files in place. Then just wait. If each register has been set up correctly, they will look on that network directory and update themselves without any touching. The update files can stay on the network without fear of being pushed again the next night because PAL checks the version number against every Ingenico every time. If an Ingenico failed to update because it wasn't turned on for example, just plug it in. Come the next "offHoursStart" it will update.

Over the years I have found that the automatic update is overkill though. Generally you don't touch things once they're working. Updating the RBAs installed on your Ingenicos is pretty rare and pushing configuration files is even rarer. If you have to update dozens of Ingenicos all at once it might make sense. But even then you're going to want to be there and see that it worked rather than come in the following morning and find chaos.

## FCC Services Control Scripts

Sometimes it is necessary to stop, start or restart the Figaro service, the Susanna service or both. A techie wouldn't have a problem with these concepts and how to accomplish them. But there's no reason to expect an average computer user to. To provide a way to make these processes as simple as possible we've provided DOS batch files (we'll call them scripts). You can find these scripts in the datafiles directory of the Windows version of the CP-EMV installation. They are also available on our DropBox. You can either place these scripts in a directory that is already part of the PATH environment variable (making them "run-able" from a command prompt or the run command) or create a desktop icon that will launch them.

Before we describe the scripts you should know that in order to run them you need to be logged in as a Windows administrator. If your account is not already an administrator account you'll need to run the script in an elevated shell. To do this from Windows Explorer right click the file and from the pop-up menu select "Run as administrator". You can right click and "Run as administrator" a desktop icon as well.

### *Figaro Service Control Scripts*

There are three scripts for controlling the Figaro service. They are:

- |                   |   |
|-------------------|---|
| StartFigaro.bat   | Starts Figaro. If Figaro isn't installed or is already running the script ends with an appropriate error message.   |
| StopFigaro.bat    | Stops Figaro. If Figaro isn't installed or is not running the script ends with an appropriate error message.  |
| RestartFigaro.bat | Stops then starts Figaro. If Figaro isn't installed the script ends with an appropriate error message. If Figaro isn't running it starts it. If Figaro is running it stops, then starts Figaro. |

### *Susanna Service Control Scripts*

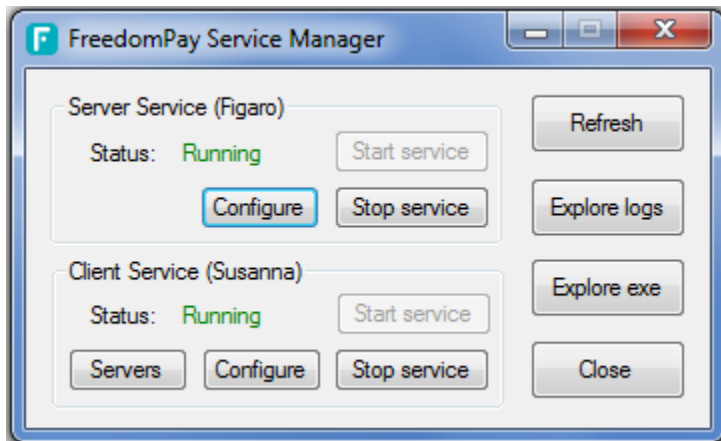
There are three scripts for controlling the Figaro service. They are:

- |                    |  |
|--------------------|--|
| StartSusanna.bat   | Starts Susanna. If Susanna isn't installed or is already running the script ends with an appropriate error message.  |
| StopSusanna.bat    | Stops Susanna. If Susanna isn't installed or is not running the script ends with an appropriate error message.   |
| RestartSusanna.bat | Stops then starts Susanna. If Susanna isn't installed the script ends with an appropriate error message. If Susanna isn't running it starts it. If Susanna is running it stops, then starts Susanna. |

## FreedomPay Service Manager

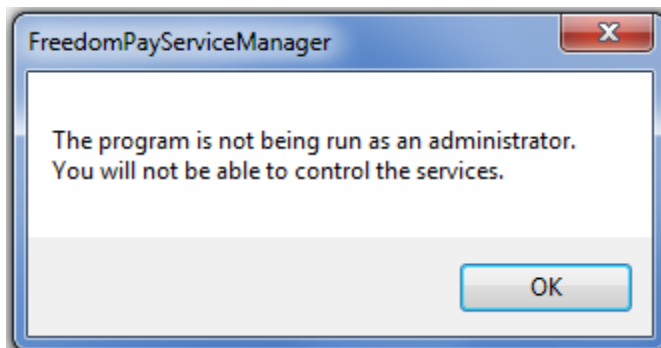
Tweaking or debugging Figaro and Susanna involves stopping and starting the services, viewing their log files and possibly editing their configuration files. And every time you make a change to either configuration file you need to restart the associated service. Add to that both configuration files have security settings that must be changed before you can save either file. You end up having two Windows Explorer sessions open and the Service control applet open to stop and start the services.

To simplify this whole process we created the FreedomPay Service Manager. It's a one-stop-shop for all these tasks. It has a single, simple dialogue shown below.



The screen is divided into two main groups of controls – one for Figaro and the second for Susanna. There are buttons for each to start and stop the service and another labeled “Configure” to open the configuration file for the associated service in NotePad. The Susanna group has an additional button labeled “Servers” to open the servers.xml file in NotePad. The right side of the screen has four buttons that don't apply to Figaro or Susanna specifically or control the application.

When you start the program you'll receive the ubiquitous Windows ACL warning screen warning you that running this program is dangerous. As usual, click “Yes” and continue on. Windows forces this upon you because this program accesses Windows services and so could allow control over important Windows functions. As the program starts it checks to see if it is being run in Administrator mode. If not you'll get the message...



The program will still start and operate, but you won't be able to stop or start Figaro or Susanna. We've added code to the program to force it into Administrator mode so we expect you'll never see this message, but if you do that's why.

Continuing on the program now checks the status of the Figaro and Susanna services. We expect to find three states – "Running", "Stopped" or "Not installed". One of these literals is loaded into the status fields in green, red or gray respectively. Depending on the status field the buttons associated with the service are enabled or disabled. Obviously it makes no sense to attempt to start a service that isn't installed or is already started. Now the program waits for you to do something.

The "Start service" and "Stop service" buttons should be self explanatory. When either is clicked the status changes to "Processing..." in gray and any other button that affects either service is disabled. We don't want the user to disrupt starting or stopping a service by starting or stopping another service or shutting down the program. The disabled buttons are enabled once the current task has finished – which may take some time. Inexplicably, sometimes Figaro and Susanna can take some time to stop or start. Usually it happens pretty quickly, but sometimes not. You can see this in the Windows services applet as well so it's not an effect of FPSM. Just be patient. It will happen when it's good and ready.

Clicking the "Configure" button will open the configuration file for the associate service. But before that the program attempts to change the permissions of the file. FreedomPay ships its configuration files with modify access turned off. So you can open the file and make changes, but you can't save it. I discussed that before in the FCC configuration section of this document. Rather than forcing you to manually changing the security settings of each configuration file this program will make the changes for you. Basically it gives the "Users" account full control over the file. Of course before editing (or changing the security settings) of any file you should always create a backup copy first. This program won't do that for you. But it's easy via the "Explore exe" button we'll discuss later. Once the file's security settings are change the configuration file is opened in NotePad.

The "Servers" button only applies to Susanna. Clicking that is exactly like clicking on the "Configure" button except the servers.xml file is opened.

The "Refresh" button merely checks the status of Figaro and Susanna and redisplay the status and enables the buttons as appropriate.

The "Explore logs" button opens a new instance of Windows Explorer already in the FreedomPay log file directory. This gives you super fast access to the logs where you can open, copy, email, zip... the files anyway you wish. It's just Windows Explorer.

The "Explore exe" button also opens a new instance of Windows Explorer, but this time you'll be in the FreedomPay executables directory. This is where the configuration files are. This makes it easy to make backups of the configuration files you're about to change.

The "Close" button shuts the program down. The red "X" button in the upper right corner of the screen does exactly the same thing.

The minimize button send the program to the system tray – not the task bar. That's nice !

## **CounterPoint Installation and Setup**

CP-EMV has a considerable footprint in CounterPoint's installation. It includes almost 90 modified programs, 14 modified files layouts and 1 entirely new data file. Setting up CounterPoint involves:

- Creating a verified backup of your existing CounterPoint installation
- Installing the modified programs and files.
- Copying the CP-EMV.REG registration file into your top-level directory.
- Initializing the new system-wide EMV information file.
- Populating several new fields that have been added within the CounterPoint software.
- Adding the new EMV fields to your forms.

Each is discussed in the paragraphs below.

### **Creating a Verified Backup of your Existing CounterPoint Installation**

Whenever you make a change to CounterPoint's programs and/or data you should create a verified backup first. Don't just assume last night's backup is enough. I can't tell you how many times I've heard from customers that discovered a serious problem that required a restore to correct only to find that the backup routine/system they had been running for years stopped working months before or in one case, had never worked at all. Make a separate backup as the system sits right now. I have my own routine for doing this, but you can do what you want. I suggest creating a directory such as syn.new right next to syn and copy (including directory structures) everything from syn to syn.new. Being empty, the TEMP directory might not be created in the backup directory. Create that. If you have your data in the default directory structures it will be backed up as well. But if you have defined separate paths for any of your packages you'll need to back them up separately.

Start CounterPoint in the backup directory and check that it actually runs. I'm going to venture beyond discussing backups here to describe my rationale. Then begin the installation into syn.new. The CP-EMV installation script requires the name of the CounterPoint top-level directory as the first parameter. So supply syn.new as that parameter. The copy will be the CounterPoint installation that gets updated. If anything goes wrong during the installation your original installation isn't affected. Once the installation is done it's easy to rename the production syn directory to syn.old and syn.new to syn. If after the installation you discover some problem that requires rolling back the original installation it's as easy as renaming syn to syn.new and syn.old back to syn. That puts you right back to running from the unmodified, production CounterPoint installation and you still have the new installation sitting in syn.new to debug and get going when you can. (But of course you'll still need to update the data files when you eventually do decide to "flip the switch".

Your regular backups might not backup the entire CounterPoint installation. They may only backup your data files and not the hundreds of CounterPoint programs that drive the software. Since this update replaces over 90 programs in several directories you need to make a backup of your programs before the installation.

## Installing the Modified Programs and Files

The bulk of the modified programs and various files associated with CP-EMV are delivered in a single zip file named cp-emv.zip. Included is a script that will copy the files to their correct directories. (The original files are renamed to \*.EMV50.) Because the installation script and command line programs differ between operating systems there are different installation procedures for Windows and Linux-based systems. Each will be described below then we'll merge back to discussing topics common to both platforms.

### Windows® Installation Instructions

To install this enhancement you must know the name of the directory in which your CounterPoint software has been installed and have some familiarity with using the DOS command line. If you do not feel comfortable with this please ask your System Administrator or your CounterPoint dealer.

This installation is not going to happen during your lunch break. Give yourself several hours to get everything installed, configured and tested.

1. As a precaution, perform a complete backup of your entire CounterPoint system (data and programs). I recommend copying the entire syn directory to another directory. Copying the files is more reliable than backing to tape or a CD/DVD. It will make it easy to access files from the backup if necessary.
2. This enhancement was delivered to you in a single zip file named CP-EMV.zip. This file must be extracted to your CounterPoint server's hard drive before you can run the installation script. The zip file has a directory structure that must be maintained. You can use any extraction tool you want, but you must make certain you retain the directory structure or the installation script will fail. Most Windows® based extraction utilities will retain this structure. However if you use an older utility such as PKUNZIP® you needed to use the -d option to force the program to extract the directory structure correctly.

We recommend you extract the files in a new directory located immediately off your CounterPoint top-level directory such as c:\syn\cp-emv. This way if you were to backup your entire CounterPoint installation (including subdirectories) you would automatically include a backup of the enhancement.

3. Have all users exit CounterPoint completely. All the way out. Not sitting at the menus.
4. The installation procedure must be run from the DOS command line. Open a Command Prompt session as an administrator.
5. Change to the drive in which CounterPoint is installed by typing
6. Change to the directory in which the enhancement files are located by typing

`x: <ENTER>` (replacing "x" by the correct drive identifier)

`CD \mods_path <ENTER>` (replacing *mods\_path* by the correct path such as \syn\cp-emv)

7. Invoke the installation script by typing the following command

```
INSTALL_EMV drive:\syn_path <ENTER> (replacing drive:\syn_path with your CounterPoint top-level directory such as C:\SYN.
```

The modified programs will be installed into their respective directories. The original programs will be saved by renaming the extensions to \*.EMV.

8. When the installation has completed, close the DOS command line session by clicking the close application button in the upper right corner of the window or by typing the command EXIT on the command line and pressing <ENTER>.

### *Unix Installation Instructions*

To install this enhancement you must know the name of the directory in which your CounterPoint software has been installed and have some familiarity with using the UNIX command line. If you do not feel comfortable with this please ask your System Administrator or your CounterPoint dealer.

This installation is not going to happen during your lunch break. Give yourself several hours to get everything installed, configured and tested.

The file cp-emv.zip contains four files.

readme	A version of this file
mods.zip	Contains the files associated with this enhancement
install_emv.sh	Script to perform the installation
unzip	Utility program to unzip mods.zip

Once you've invoked the installation, it should be allowed to run to completion or your system may be left in an unstable state. If you feel the installation MUST be interrupted immediately you may press the <DEL> key to do so. The installation script does not alter any data files. It only copies the modified programs to their respective directories.

1. As a precaution, create a complete back-up of your CounterPoint system (data and programs).
2. Log in as root.
3. Copy the cp-emv.zip file to your CounterPoint top-level directory.
4. Navigate to your CounterPoint top-level directory. For example, type:

```
cd /syn <ENTER>
```

Replace "syn" with the name of your CounterPoint top-level directory. You should now be at the UNIX '#' prompt in your CounterPoint top-level directory.

5. Unzip cp-emv.zip into the top-level directory. You should already have the unzip utility there since it is distributed as part of the Unix CounterPoint installation.
6. Run the script 'install\_emv.sh' to perform the actual installation by typing:

```
install_emv.sh <ENTER>
```

The modified CounterPoint programs will be installed into their respective directories. The original programs will be saved by appending their extensions with \*.EMV.

Once the installation is complete you'll find two files in your top-level directory named emvpas.linux and emvpas.SCO. Copy the version that is appropriate for your system to the name emvpas (no file extension).

One of the files copied to your CounterPoint top-level directory during the installation process is a script named 'uninstall\_emv.sh'. It allows you to remove the modified programs and restore the original programs if necessary. To run it, navigate to your CounterPoint top-level directory and type 'uninstall\_emv.sh' (no quotes) and press the <ENTER> key. Note that uninstall\_emv.sh only removes the modified programs and restores the original programs. If you made any changes to your data files after installing the modification, running uninstall\_mods.sh will not back out the changes to your data files.

### *Back to Common Installation Instructions*

At this point your CounterPoint system will now have been updated with the modified programs and files, but you still need to set up quite a few things.

### **CP-EMV.REG**

The file CP-EMV.REG carries the activation key for your system. It was provided to you separately when you purchased CP-EMV. Copy this file to your CounterPoint top-level directory. It is not portable from one CounterPoint installation to another. Its value is tied to (among other things) your CounterPoint serial number, number of users and system type. When you purchased CP-EMV you provided us a screenshot of your registration screen which we used to generate the key. So you cannot copy this file to another installation and expect it to work. Likewise, you cannot change any of these fields in your system as that will invalidate your CP-EMV activation key. If you plan to change your number of users or system type (or anything on the screen) you need to notify us first because once you make the change EMV credit card processing will be deactivated.

Note that you can add comments to the file without affecting its function. This might be handy if you need to document details about which installation the key is for, when it was installed and so on.

Below is a sample file.

```
#####  
#####          Infinity Software Solutions          #####  
#####  
##### This file contains the registration key for our add-on product #####  
#####          EMV CC Processing          #####  
#####          -----          #####  
##### The only required line in this file is the line that carries the #####  
##### registration key. The line is formatted REG-KEY=XXXXXXXXXXXXXXXXX #####  
##### where XXXXXXXXXXXXXXXXXXXX is the 15-digit registration key. The #####  
##### registration key (if valid) will allow the add-on to run in demo #####  
##### or production mode. Demo mode will allow the enhancement to #####  
##### operate in production mode until a predetermined date. This #####  
##### will give the client sufficient time to set the system up and #####  
##### test the results before expiring. #####  
#####          -----          #####  
##### Note that you can add additional text to this file as long as #####  
##### the first 8 characters of the line is not "REG-KEY=". #####  
#####  
##### Below is a demo registration key (07/31/21) =====  
REG-KEY=48G8CB7D4BG8072VEIN7HBOABCD  
#####
```



## Upgrading CP-EMV

There are really only three reasons to upgrade CP-EMV; when you're upgrading to a newer version of FreedomPay's software, to install new features in CP-EMV or to correct a bug you're experiencing. If your system is working fine and there's no impetus to make changes don't make changes.

FreedomPay continuously updates their FCC software. But you don't need to chase these updates. Unless their update is mandatory you can stay on the version you're on. If you do decide to upgrade your FCC you should check that CP-EMV actually supports the new version.

There are other considerations. The Ingenicos have their own software (called RBAs) as well. Sometimes these RBAs have bugs or are lacking a feature you might want. In that case the RBA can be upgraded separately from the FCC and CounterPoint. But care must be taken. Some RBA packages have minimum FCC version prerequisites. If you upgrade an Ingenico to a version that your version of the FCC cannot support you might cause more problems than the desired feature provided or even break something.

Like FreedomPay, we are also updating CP-EMV. Generally speaking, you can install a CP-EMV update without breaking anything. Our updates don't drop support for older versions of the FCC. CP-EMV has nothing to do with the RBAs running on the Ingenicos.

Since all programs have changed in some way from CP-EMV version 3.x through the current version we're not going to archive the changes these upgrades provided in this document. We have a separate document "CP-EMV Release Notes" for that.

### **Installation**

When we modify CP-EMV to support an FCC upgrade it usually requires updating quite a few programs and typically involves core programs. So we compile all programs at the same time regardless of whether the program actually needed to be recompiled. This assigns all of the programs associated with a base date stamp. This simplifies identifying (both here and in the field) how-old/what-version of the software each program belongs to.

So when you upgrade CP-EMV you actually reinstall the software as if it was the first time. There is no separate upgrade process other than the installation process.

**As with ANY installation or upgrade we highly recommend you create a complete backup of CounterPoint (programs and data) before you begin.**

## Significant Changes From Version to Version

First, understand that our versions are based on upgrades of FreedomPay's FCC software. We actually number our versions after (approximately) their FCC equivalents. So we won't create a new version of CP-EMV unless there is a matching upgrade to the FCC that forced it. The sections below only mention changes between the versions that require some configuration change. There are numerous – sometimes significant – changes between versions that we haven't listed below. If you want more a detailed listing of CP-EMV's modification history look at the separate Release Notes document.

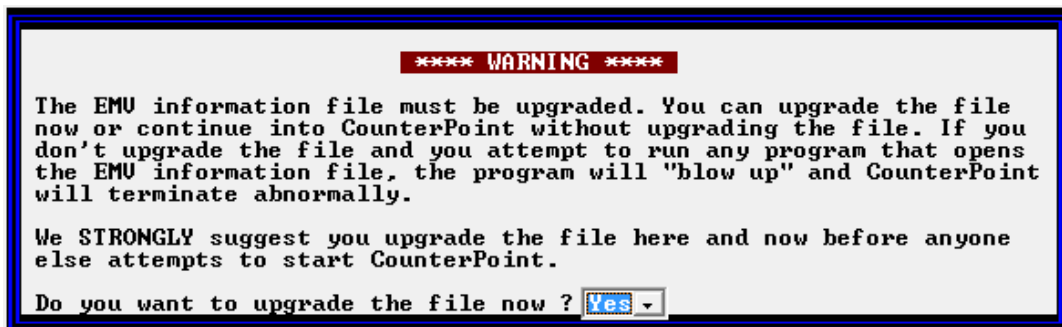
### Version 3.x to Version 4.0

Version 3.x of CP-EMV used a file named CP-EMV.INI in the top-level directory to provide the information necessary to connect to Figaro. This methodology is simple and worked just fine, but created a bottleneck for large installations with too many stores to upgrade all at once. A better solution for large CounterPoint installations would be to have two Figaros, one 3.x and one 4.0 with different IP addresses. Then each store could point their authorization requests to the Figaro running the version of the software they require.

Version 4.0 of CP-EMV obsoletes the use of the CP-EMV.INI file and moves those connection parameters to a new screen in [Setup>Point of Sale>Stores>"Configuration options"](#). We collected all of the fields involved with EMV credit card processing (at the store level) and created a new screen in that program. **Once the upgrade has been performed you will need to manually enter these connection parameters in [Setup>Point of Sale>Stores>Configuration options](#) for each store. This is very important !** Alternatively, you might run a new program we created for this release that verifies/updates the CP-EMV control fields in your store configuration records. You can see the details on that program [HERE](#).

With the introduction of version 4.0, came the need to define which version a store is running. So we added a new "FCC Version number" field to the store configuration screen. All stores need to set this new field to the version they are running even if they are still using version 3.0.

Note that we changed the format of the EMV information file in a substantial way for version 4.0. So the file must be reformatted after the upgrade. There is no separate menu selection to perform this conversion. You do not need to export the file before upgrading (as was common in CounterPoint upgrades). The first time you start CounterPoint after installing the new software CounterPoint will recognize the EMV information file is in the old format and you'll be presented the screen below.



As the message states, we STRONGLY suggest you upgrade the file immediately. There's really no reason not to. You (and any other less capable person) could ignore the message, proceed into CounterPoint and blow up in any of the 24 programs that open the EMV file. So do yourself a favor and just allow the program to convert the file now. As it converts the file it displays a records count (it runs so fast I am unable to grab a screenshot for this document).

As mentioned, this upgrade provides the ability to store signature capture files locally. There's an entire section describing this feature towards the top of this document. Refer to that section for a full description.

If you had set up sounds to play during the authorization process you will need to change the names of the environment variables. Version 3.x used the variables GOOD\_AUTH\_WAV, BAD\_AUTH\_WAV and PARTIAL\_AUTH\_WAV to carry the names of the wave files to be played indicating the result of the authorization attempt. Those environment variables have been renamed to GoodAuthWav, BadAuthWav and PartAuthWav respectively.

#### *Version 4.0 to Version 4.1*

We added a new version number to the list of allowable values for the field "FCC Version number" on the store configuration screen. Each store that upgrades to the new version of the FCC needs to update this field to match.

#### *Version 4.1 to Version 4.2*

We added a new version number to the list of allowable values for the field "FCC Version number" on the store configuration screen. Each store that upgrades to the new version of the FCC needs to update this field to match.

Version 4.0 of CP-EMV had the field "Auto-update customer cards" in the store configuration record. But that created a psychotic relationship between customers and stores. The customer file is common to all of the stores within a CounterPoint installation. But by having this field on the store record it created a situation where one store would update a customer's cards, but not another even though to the customer, both stores are the same company. So we moved the field to the customer cards definition screen in [Setup>Customers>Control](#).

#### *Version 4.2 to Version 5.0*

We added a new version number to the list of allowable values for the field "FCC Version number" on the store configuration screen. Each store that upgrades to the new version of the FCC needs to update this field to match.

We made significant changes to the Customer Cards list/purge program. We expanded the types of cards that can be reported and added the ability to identify suspicious customer card records.

We added the ability to enter customer cards to the POS and O/E add a customer on-the-fly programs.

We added totals to the EMV Transaction History and EMV Reconciliation reports.

We created the FreedomPay Services Manager utility to assist configuration and maintenance.

We created two Susanna configuration file upgrade utilities.

## Updating CP-EMV

The terms “upgrade” and “update” are often used as synonyms, but they are quite different in the software world. An upgrade takes software to a newer version. That’s means an incremental change to the version’s major version number. It often implies a change that makes the new version somehow incompatible with the prior version. Thus an upgrade is a non-trivial process. An update merely refines the same version. It might increment the most insignificant digit of the version number.

When you first installed your current version of CP-EMV the package included all of the latest versions of the programs. This is because we keep the installation package updated continuously. Over time, as we add and change programs your installation will become “stale”. This isn’t necessarily bad though. If you don’t need any of the changes to the software you don’t need to install them. But if you choose to update CP-EMV you have two choices. You can reinstall the complete package which will have all of the latest versions of the programs or you can install the update package which replaces just those programs that have changed since your version was first released.

As we make changes to a version we add these changes to the installation package as well as a separate update package. The packaging and procedures for installing an update are the same as installing the initial package. The only difference is the command that invokes the update (update\_EMV) and the size of the package which is considerably smaller. So install an update download the update package and follow the directions for installing the CounterPoint modifications. But rather than running install\_EMV to kick of the process us the command update\_EMV.

When you first installed your current version of CP-EMV the package included all of the latest versions of the programs. This is because we keep the installation package updated continuously. Over time, as we add and change programs your installation will become “stale”. This isn’t necessarily bad though. If you don’t need any of the changes to the software you don’t need to install them. But if you choose to update CP-EMV you have two choices. You can reinstall the complete package which will have all of the latest versions of the programs or you can install the update package which replaces just those programs that have changed since your version was first released.

As we make changes to a version we add these changes to the installation package as well as a separate update package. The packaging and procedures for installing an update are the same as installing the initial package. The only difference is the command that invokes the update (update\_EMV) and the size of the package which is considerably smaller.

As the update replaces programs and files it creates a backup of each file it changes. The original copy of the file is renamed by appending “.50x” to the file name where “x” is an incrementing alphabetic character. So the first update renames the file to \*.50a, the second to \*.50b and so on. This allows retaining a copy of all of the versions of each file CP-EMV changes.

## **Upgrading FreedomPay**

The software to upgrade your FCC components can be found on our DropBox. There are several versions. When FreedomPay offers them, we provide different flavors of each that allow installing the full FCC package, just the client software or possibly even a version that upgrades your installation from a prior version.

The Full installation contains all of the files you'd need to install Figaro, Susanna, the FCC Test Client and the FCC User Interface for the first time including all the associated libraries and required .Net Framework. The Full version is overkill if you are merely upgrading an existing system. The Client Only installation has just Susanna and the FCC User Interface and all the associated libraries and .Net Framework. The Upgrade Only installer doesn't contain the associated libraries and .Net Framework. That makes the file smaller and the installation simpler. It assumes you already have a prior version of the FCC installed and working. This is the installation option we chose when we upgraded our machines here.

### **Upgrading Figaro**

You probably only have a single installation of the FCC server (Figaro) serving your entire enterprise so you'll only need to do this once. Obviously, if you have multiple Figaros running you'll need to perform this on each.

The process to perform the upgrade is straightforward and simple. Copy the installation file to the target machine, in Windows Explorer right-click the executable file, click "Run as administrator" and follow the prompts.

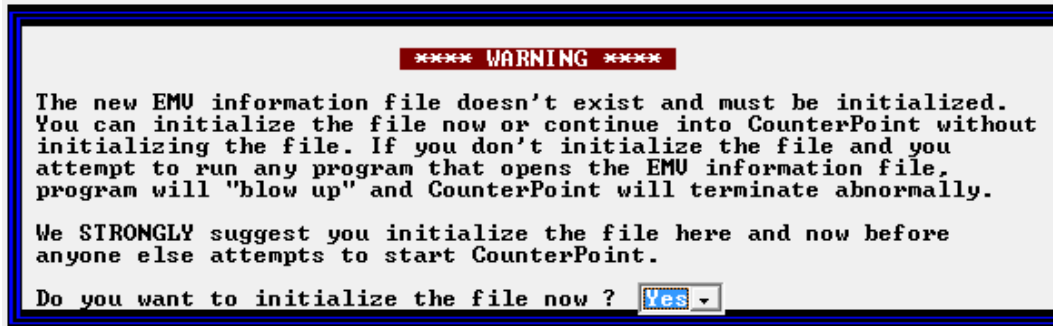
### **Upgrading Susanna**

Each register is running a Windows service component nicknamed Susanna. Once you upgrade Figaro you have to upgrade the Susanna(s) that communicate with it. If FreedomPay released an upgrade package run that. Copy the installation file to the target machine, in Windows Explorer right-click the executable file, click "Run as administrator" and follow the prompts. It likely won't have any parameters and will do all the work for you. Otherwise you'll need to download the full FCC installation package and "install" that. (The instructions on that have already been covered.) Just be sure to install only those FCC components that are part of the Susanna setup.

## Modifications to CounterPoint

### The New EMV Information File (SYEMVF)

This enhancement adds a new file to CounterPoint to carry the additional EMV information for transactions. The file is named SYEMVF and it resides in the SYDATA directory. If you attempt to start CounterPoint and this file doesn't exist you'll receive the message:



As the message states, we STRONGLY suggest you initialize the file immediately. There's really no reason not to. You (and any other less capable person) could ignore the message, proceed into CounterPoint and blow up in any of the 24 programs that open the EMV file. So do yourself a favor and just allow the program to initialize the file now. It happens so fast you won't even notice. You'll receive this message every time you start CounterPoint until you do.

Let's say you want to initialize the file manually. Navigate to *File Utilities>System* and select "4. Initialize files". Answer "N" to every file except for "10. EMV information file". Do I need to remind you that if you answer "Yes" to any other file the information in that file will be immediately and permanently erased? So before you press ENTER at FNTC look back at the values you've entered just to be certain that only the EMV Information file is selected. Press ENTER and the file will be created. Do this for EVERY company that is running in that CounterPoint installation even if they aren't using EMV. The modified CounterPoint programs will still want to open the file and if it's not there CounterPoint will fail with file error 35 on the missing file.

The contents of the file are discussed later in this document.

### Messages Displayed During Transactions

All programs that obtain authorizations (or tokens) now display a standardized message on the screen during the process. Once CounterPoint calls out for the authorization the program displays a message similar to "Waiting for tender & authorization". To the right of the message begins a countdown in seconds informing the clerk just how long he should wait for a response. Some of our customers have their timeouts set to crazy long settings and the clerks would give up too early and restart CounterPoint. The countdown should tell the clerk to wait at least until the timeout.

If a timeout does occur the program asks the clerk "Continue waiting?" The clerk can answer "Y" and wait some more or answer "N". The software then sends a special "cancel" command to (hopefully) prevent FreedomPay from processing the authorization later.

## **Setup>System>Company**

Actually, we didn't change anything here. We just wanted to remind you that if all of the stores running in a CounterPoint installation have been converted to using CP-EMV (as opposed to CP-Gateway) you can disable EDC in CounterPoint and go back to using normal (sane) password policies.

If you have been using CP-Gateway you invariably had to set a value for the environment variable "WINEDC". That variable does more than control credit card processing in CounterPoint. It also turns on the need for your passwords to conform to PCI DSS standards. If all of your stores are using CP-EMV you no longer need to meet that standard since CounterPoint is "out of scope". Set the variable to "WINSTD" or remove it completely and the password maintenance routines will no longer require you enter PCI-compliant values. No more changing passwords every 90 days !

## **Setup>Point of Sale>Control**

We based this enhancement on another, much older product we had written that performs level-3 credit card processing. Part of that mod was to allow placing the ticket's authorization codes in the header record's profile fields. That's a bit redundant since the authorization code is already stored in the tender's reference-2 field and also in the new EMV information file, but we didn't see any reason not to include it anyway. There are reports and/or forms that display the profile fields and not the reference fields so it could serve a purpose for someone.

When a ticket is authorized the program looks to see if one of the ticket profile fields has been defined to carry the auth code for that tender (as in tenders 1, 2 or 3). It makes this determination by looking at the tag that's been defined for the profile fields. If tender-1 was an EMV tender type and one of the profile fields is tagged "CC Auth code-1" (spelling and case matter) the program will copy the authorization code the header's profile 1 field as well. The same is true for tenders 2 and 3 and profile field tags "CC Auth code-2" and "CC Auth code-3". If you don't want to retain authorization codes in any of the profile fields just don't define any of the profile codes with the "CC Auth code-#" tag. The fields are independent of each other so you could define "CC Auth code-1" but not the other 2 to allow you to retain just the authorization code for the first tender.

## **Setup>Point of Sale>Stores>Stores**

In "vanilla" CounterPoint, the software controlled the entry of the credit card information and could see the unencrypted credit card data. That's good if you want the total control over the process, but it is also the huge liability that most merchants are trying to shed. As discussed before CP-EMV hides this information from CounterPoint. With CP-EMV CounterPoint merely requests payment of a particular amount and the customer determines and provides the tender type. So there's no need for the clerk to ascertain which type of tender is being offered and enter the correct pay code for that tender type. In fact the clerk doesn't need to (and shouldn't) handle the customer's credit card at all. The end result of this is there is no longer a need to define a separate pay code for each type of credit card, debit card, VISA gift card and whatever. You only need a single pay code for all of these – "EMV". FreedomPay's statement to you will separate your payments by tender so you will still get the totals you need at the end of each month.



But if you do want to define a separate tender code for each type of electronic tender you can. You can still assign each a separate G/L account number and that will all still work the same way it did before. But if you allow the customer to perform the swipe/insert/wave your clerk will first have to ask your customer what type of card they are using in order to enter the right pay code for that card. Of course you can assume that they'll get it wrong now and then.

EMV pay codes are defined a little differently than non-EMV. The primary difference is you don't want to define validation fields for EMV pay codes because you don't want to be prompted for additional data during the tender process. So for each store in *Setup>Point of Sale>Stores>Stores* go to the third screen and add a pay code for EMV payments. You can use any code value you want since the enhancement doesn't actually use the code to determine what's an EMV pay code and what isn't. Give it the description you desire and assign it the type "Credit crd". **Leave the Validate fields blank.** Of course "OpnDrw" and "Chg" should be set to "N". Enter the account number you desire. Note that you might need to get your accountant involved a bit if you are currently using different account numbers for each credit card processor.

Here's a sample. See pay code 17.

CounterPoint 7.5  
7.5.18

Stores  
Store: 2 Main Street

PayCode	Desc	Type	OpnDrw	Uvalidate-1	Uvalidate-2	Chg	NoCnt	Acct-#
1.	Cash	Cash	Y			Y	N	1010
2.	Chk	Checks	Y	DL#		Y	N	1010
3.	MC	MasterCard	Y	Cred Card#	Exp.Date	N	N	1040
4.	Debt	Debit card	N	DL#		N	N	1010
5.	Coup	Coupon	Y			Y	N	1510
6.	Visa	Visa	N	Cred Card#	Exp.Date	N	N	1040
7.	Disc	Discover	N	Cred Card#	Exp.Date	N	N	1040
8.	AmEx	Am Express	N	Cred Card#	Exp.Date	N	Y	1040
9.	Food	Food stamp	Y			Y	N	1010
10.	Stcr	Store crdt	N	Store card		N	N	7500
11.	Gift	Gift cert	Y	Gift cert#		Y	N	7500
12.	A/R	A/R charge	N	DL#		N	N	1200
13.	CkCd	Check Card	N	DL#		N	N	1010
14.	Pts	Points	N			N	N	7500
15.	AMEX	AmExpress	N	Cred Card#	Exp.Date	N	N	1040
16.	UISA	Uisa	N	Cred Card#	Exp.Date	N	N	1040
17.	EMU	EMU Card	N			N	N	1040
18.	SUC	SUC	N	Card-#		N	N	2080

Field number to change ?

Create a single entry for EMV card tenders. This one entry will handle all of your EMV processing. Remember, the clerk isn't handling the card anymore. He doesn't know what type of card is being tendered. It could even be a Givex gift card. The clerk wouldn't know which pay code to select at the time of the sale. This is why we created the [EMV Reconciliation report](#). It allows you to figure out how much each card type brought you even though they were all processed using the same pay code account number.

You'll probably want to remove any other credit card type tenders from this screen lest your employee's continue to use them in error.



## Setup>Point of Sale>Stores>Configuration options

Each store has its own FreedomPay configuration. In fact, each store within a single installation of CounterPoint can be defined to use FreedomPay's credit card processing or CounterPoint's built-in credit card processing. **Each store must be configured to use either FreedomPay or CP-Gateway.** We cannot stress this enough.

Enter the store number and go to the new 9<sup>th</sup> screen titled "Credit Card Processing options". Field "1. Credit card gateway" is the new field to be set up. It allows one of two possible values – "CP-Gateway" or "FreedomPay". If you choose "CP-Gateway" you're done. You don't have to enter any of the other fields. Of course you won't be able to authorize any credit cards within CounterPoint since NCR disabled CP-Gateway for CounterPoint. If you select "FreedomPay" you'll need to enter the other fields as well.

Here's a sample screen.

```
Configuration options 7.5.20
Store number: 1      Main Street

      --- Credit Card Processing options ---

1. Credit card gateway      FreedomPay
2. FCC Uersion number      4.2x
3. FreedomPay store-id     1418014016
4. FreedomPay terminal-id  2418339013
5. Floor limit             50
6. Store signatures locally ? Y
7. FCC Server IP address   127.0.0.1
8. FCC Server port        1012
9. FCC Server timeout <in seconds> 150
10. Operating mode        Normal

Field number to change ? 
```

If your screen doesn't look like this you're not running version 4.2 of CP-EMV. Download your version from our DropBox and read that version of this documentation or upgrade CP-EMV to the 4.2 version.

**You must set these fields for each store in your enterprise.** If you don't specify which process to use the result could be benign or disastrous. The reason is quite simple. If you don't provide a value the flag will default to what programmers term "undefined". That means whatever value is already in that record position. If that value is an "F" (FreedomPay) or a "C" or a space (CP-Gateway) you're in luck. Any other value is meaningless to the programs and will cause you great difficulties (like not getting paid). In the code there are many places where the program asks "If this store is using FreedomPay do this, else do this" or visa-versa. Other places may only ask "If using FreedomPay". If the store isn't set up to be either the results are completely undefined, but expectedly bad.

Setting this field is so important I'm going to say it a third time – **set this field for every single store in your CounterPoint installation.**

The field “**2. FCC Version number**” determines what format of authorization request to build. If you get this wrong you’ll either never get an authorization or CounterPoint will hang after FreedomPay authorizes the request. There are four possible values (at this time) – “3.x”, “4.0x”, “4.1x” and “4.2x”. You should select the value appropriate for the version of the FCC you have installed. The four versions are incompatible with each other. For example, you cannot run a version 3.x Figaro with a 4.0x Susanna (or visa versa).

FreedomPay provided values for the fields “**3. FreedomPay store-id**” and “**4. FreedomPay terminal-id**” when you boarded. Enter those values here.

The “**5. Floor limit**” field determines the maximum amount you’re willing to authorize off-line. As discussed previously in this document offline transactions are optimistically approved as long as the amount is below this value. Later, when your Internet connection is reestablished Figaro will begin attempting to authorize these offline transactions. There is the possibility that one or more may fail to authorize in which case you won’t get paid. You’ll need to decide if you want to employ offline authorizations and if so, what threshold of liability you’re willing to accept.

The value zero in this field has different meanings between FCC versions 3.x and later versions. In version 3.x a zero here means “Use the floor limit value in the Figaro configuration file”. In version 4.0 and above a zero means “No offline authorizations are allowed”.

The field “**6. Store signatures locally ?**” controls if you store customer signatures on your CounterPoint server. Answering “N” doesn’t prevent signature data from being collected by FreedomPay. Regardless of how you answer this parameter FreedomPay will still repository the signatures on their server. You can access them via their Enterprise Portal. See the section “Signature Capture” towards the top of this document for more information on this feature of CP-EMV.

The field “**7. FCC Server IP address**” provides CP-EMV the Figaro IP address to send the authorization requests to. It cannot be left blank. It can be local IP address (such as 192.168.1.55), an external address (such as 47.199.166.14) or even the “loop-back” address (127.0.0.1) if Figaro is running on the same machine as CounterPoint.

The field “**8. FCC Server port**” tells CP-EMV the Figaro port number to send the authorization requests to. It should have the same value as Figaro’s NVPport parameter value. The default value is “1012”.

The field “**9. FCC Server timeout**” tells CP-EMV how long to wait for a response back from Figaro. The default value is 150 seconds, but you can enter any 3-digit value. CounterPoint calls a separate command line program to perform the communication and starts counting the seconds as it waits for the response. If the response doesn’t arrive before this timeout setting CounterPoint will assume something failed, will display an error message and will prompt for the tender again.

If you change any of the timing settings from the recommended values you need to carefully add up the round trip timings for CounterPoint to Figaro, Figaro to Susanna, waiting for the swipe, the auth request, waiting for the signature (if required), Susanna back to Figaro and Figaro back to CounterPoint. One hundred and fifty seconds might seem excessive, but once you add all these segments together you can see they add up pretty quickly.

The field “**10. Operating mode**” controls how CP-EMV interacts with the FreedomPay software. “Normal” mode is the usual setting (thus the name). This is when each register has Susanna installed and controlling the Ingenico via a USB cable. “Multilane” mode is when the store has IP-based Ingenicos. In this configuration all of the Ingenicos Ethernet cables are attached to the store’s network switch and there’s a single Windows machine running a single instance of Susanna.

Normally, Figaro receives the authorization request from CounterPoint, identifies the workstation-id then routes the request to the associated Susanna for that register. In a multilane mode environment when Figaro receives the request there’s only a single Susanna to which to send the request. The Susanna has a table loaded inside of it that allows it to route the request to the correct Ingenico. The Susanna handles the request in the same manner as if the Ingenico was attached via a USB cable, but it handles it via the network and it handles all of the Ingenicos.

Obviously this is an important setting and nothing will work if it is set to the wrong value.

We’ve had an issue with Synchronics for some time and we’re taking ownership over the issue. They’ve had a typo on a screen in this program since version 7.5.18 and we fixed it ! “roundng” is now “rounding”.

```
Configuration options 7.5.20
Store number: 1    Main Street

      --- Ticket entry options 3 ---

1. Display cost during item zoom           None
2. Display cost during price override      None
3. Display total ticket cost               N      Tot qty  N
4. Allow non-inven item cost changes ?    N
5. Allow drop-ships ?                      Y
   Allow drop-ship cost changes ?         Y
6. Minimum profit percent                  0
   Minimum price level                     None
7. Override price by profit percent ?      Y      Cost basis  Average
8. Use miscellaneous charges ?             Y      Account #  8410
   Miscellaneous charges text              Freight  Tax ?  N
   Use calculated freight amount ?         N
9. Use customer discount percent ?        Y
10. Apply ticket discounts to              None
11. Discount rounding                   Normal

Field number to change ? 
```

## Setup>Point of Sale>Forms

### New EMV Tender Fields

EMV processing adds a few new credit card related fields that you may want to print on the receipt or use to control fields that are printed on the receipt. As mentioned earlier, CP-EMV supports partial payments and so needs to have the ability to print the card's remaining balance on the receipt. To handle these requirements we've added 63 new fields for each of the 3 tenders (that's right – 189 new tender fields in total!) to the header/footer fields in the POS forms designer. That's a lot and frankly I don't know why someone would want to use most of them. But they are included with the authorization response and I've personally seen at one receipt that printed most of them. Use them as you see fit (if at all).

Below are the fields for the first tender. The other two tender's fields the same, but labeled "Pay 2:" and "Pay 3:".

<u>Field name</u>	<u>Field type</u>	<u>Length</u>
Pay 1: EMV appr amt	Numeric	9 integers 2 decimals
Pay 1: EMV auth code	AlphaNumeric	8
Pay 1: EMV bal rem	Numeric	7 integers 2 decimals
Pay 1: EMV card type	AlphaNumeric	10
Pay 1: EMV CVM	AlphaNumeric	5
Pay 1: EMV CVM meth	AlphaNumeric	15
Pay 1: EMV date	Date	8 integers
Pay 1: EMV DCC acppt	AlphaNumeric	1
Pay 1: EMV decision	AlphaNumeric	1
Pay 1: EMV entry mod	AlphaNumeric	10
Pay 1: EMV error cod	Numeric	6 integers
Pay 1: EMV exp date	AlphaNumeric	4
Pay 1: EMV floor lim	Numeric	9 integers
Pay 1: EMV issuer	AlphaNumeric	20
Pay 1: EMV lane-id	Numeric	10 integers
Pay 1: EMV masked CC	AlphaNumeric	20
Pay 1: EMV nam on CC	AlphaNumeric	30
Pay 1: EMV offline	AlphaNumeric	1
Pay 1: EMV part amt	Numeric	9 integers 2 decimals
Pay 1: EMV pmt flag	AlphaNumeric	1
Pay 1: EMV ref code	AlphaNumeric	20
Pay 1: EMV req amt	Numeric	9 integers 2 decimals
Pay 1: EMV req ID	AlphaNumeric	32
Pay 1: EMV sig avail	AlphaNumeric	1
Pay 1: EMV TAC-DEN	AlphaNumeric	10
Pay 1: EMV TAC-DFLT	AlphaNumeric	10
Pay 1: EMV TAC-ONLIN	AlphaNumeric	10
Pay 1: EMV TAG-50	AlphaNumeric	20
Pay 1: EMV TAG-5F2A	AlphaNumeric	5
Pay 1: EMV TAG-5F34	AlphaNumeric	5
Pay 1: EMV TAG-82	AlphaNumeric	5
Pay 1: EMV TAG-95	AlphaNumeric	10
Pay 1: EMV TAG-9A	AlphaNumeric	10

Pay 1: EMV TAG-9C	AlphaNumeric	10
Pay 1: EMV TAG-9F02	AlphaNumeric	10
Pay 1: EMV TAG-9F03	AlphaNumeric	10
Pay 1: EMV TAG-9F07	AlphaNumeric	5
Pay 1: EMV TAG-9F0D	AlphaNumeric	10
Pay 1: EMV TAG-9F0E	AlphaNumeric	10
Pay 1: EMV TAG-9F0F	AlphaNumeric	10
Pay 1: EMV TAG-9F10	AlphaNumeric	15
Pay 1: EMV TAG-9F12	AlphaNumeric	20
Pay 1: EMV TAG-9F1A	AlphaNumeric	5
Pay 1: EMV TAG-9F26	AlphaNumeric	20
Pay 1: EMV TAG-9F27	AlphaNumeric	5
Pay 1: EMV TAG-9F34	AlphaNumeric	10
Pay 1: EMV TAG-9F36	AlphaNumeric	5
Pay 1: EMV TAG-9F37	AlphaNumeric	10
Pay 1: EMV TAG-AID	AlphaNumeric	15
Pay 1: EMV TAG-ARC	AlphaNumeric	5
Pay 1: EMV TAG-DF03	AlphaNumeric	10
Pay 1: EMV TAG-DF05	AlphaNumeric	10
Pay 1: EMV TAG-DF05	AlphaNumeric	10
Pay 1: EMV TAG-IAD	AlphaNumeric	15
Pay 1: EMV TAG-TSI	AlphaNumeric	5
Pay 1: EMV TAG-TVR	AlphaNumeric	10
Pay 1: EMV time	Time	8 integers
Pay 1: EMV tip amt	Numeric	9 integers 2 decimals
Pay 1: EMV token	AlphaNumeric	32
Pay 1: EMV token exp	AlphaNumeric	4
Pay 1: EMV trx type	AlphaNumeric	1
Pay 1: EMV trx-id	AlphaNumeric	20
Pay 1: EMV void req	AlphaNumeric	32

I don't know what the purpose of many of these fields is. So I'll just describe the purpose for the ones I do know.

Pay 1: EMV appr amt	The amount that was approved. Note this may not be the amount requested.
Pay 1: EMV auth code	The authorization code. Returns and voids don't return an auth code.
Pay 1: EMV bal rem	Balance remaining on a debit card.
Pay 1: EMV date	The date the transaction was created.
Pay 1: EMV card type	In our testing we've seen "credit" returned.
Pay 1: EMV decision	Whether the auth worked or not. Expect an "A" in this field.
Pay 1: EMV entry mod	"icc" for sales. Blank for voids and returns.
Pay 1: EMV error cod	Returns 100 if successful. Otherwise it indicates the error encountered.
Pay 1: EMV exp date	The card's expiration date.
Pay 1: EMV floor lim	The floor limit in effect during the transaction.
Pay 1: EMV issuer	The card name such as "VISA".
Pay 1: EMV lane-id	The lane-id for multilane installs. Expect a "0" for this.
Pay 1: EMV masked CC	The card's masked number.
Pay 1: EMV nam on CC	The cardholder's name as it appears on the card.
Pay 1: EMV offline	Flag indicating the transaction was an offline authorization.
Pay 1: EMV part amt	The amount authorized during a partial authorization.
Pay 1: EMV pmt flag	Expect a "Y" if the card was processed via CP-EMV.

Pay 1: EMV ref code	“sss-rrr-tttttt” where “sss” = store, “rrr” = register and “ttttt”=ticket.
Pay 1: EMV req amt	The amount initially requested.
Pay 1: EMV req ID	Unique value used to identify the transaction.
Pay 1: EMV sig avail	Is a signature captured during the transaction ? (Y/N).
Pay 1: EMV time	The time the transaction was created.
Pay 1: EMV token	A unique value that uniquely identifies the customer’s card.
Pay 1: EMV token exp	The date the token above expires.
Pay 1: EMV trx type	“S” = Sale, “R” = Return, “V” = Void.
Pay 1: EMV trx-id	A less unique transaction identifier.
Pay 1: EMV void req	The request-id of the transaction that was voided during a void.

## Enhanced Form Field Process

While adding the EMV fields to the receipts we discovered there was something simple missing from the POS forms designer – the ability to simply push the rest of the fields in a group up or down one line. When you key the line number (but before you key the column) the program offers <F2> to “Insert line”. If you answer “Y” the program then asks “Ok to move remaining fields to new line?”. If you answer “Y” the program does push the remaining fields in the group down by one line. But it doesn’t ask the opposite question when you delete a field. The line the field was assigned becomes a “hole” in the form designer. To get rid of the missing line you have to manually change each below the missing line assignment. And you’d better know exactly how many lines you need to adjust every line in the group (the numbers may be different if you’re making several changes) or you’ll have to repeat the process (again and again).

Does that matter so much? Probably not to most people. But we’re Infinity Software Solutions and we have higher standards. What we found most cumbersome is you can’t make these changes easily from FNTC while using the <F1> and <F2> keys to navigate through the fields. It’s simply far easier to <F1> through the fields, spot where a field needs to be added or missing lines adjusted without leaving where you’re at on the screen.

We added code to the program to allow it to “Bump down” all of the fields on the lines following the current line by one by pressing <F5> at FNTC. If the form is paginated there’s a limit to the number of lines a particular group can have. The program first looks to see if the group has room to bump the remaining fields in the group down one line. If there’s room the program offers <F5>.

We added code to allow the program to “Pull up” all of the fields on the lines following the current line by one by pressing <F6> at FNTC. The program first looks to see if there’s at least one unassigned line number between the current line and the next field’s line in the group. If there is the program offers <F6> to pull those lines up a line and fill the gap.

The <F5> and <F6> keys are only offered at FNTC when you’re in “change mode” - not if you’re adding a new field to the form.

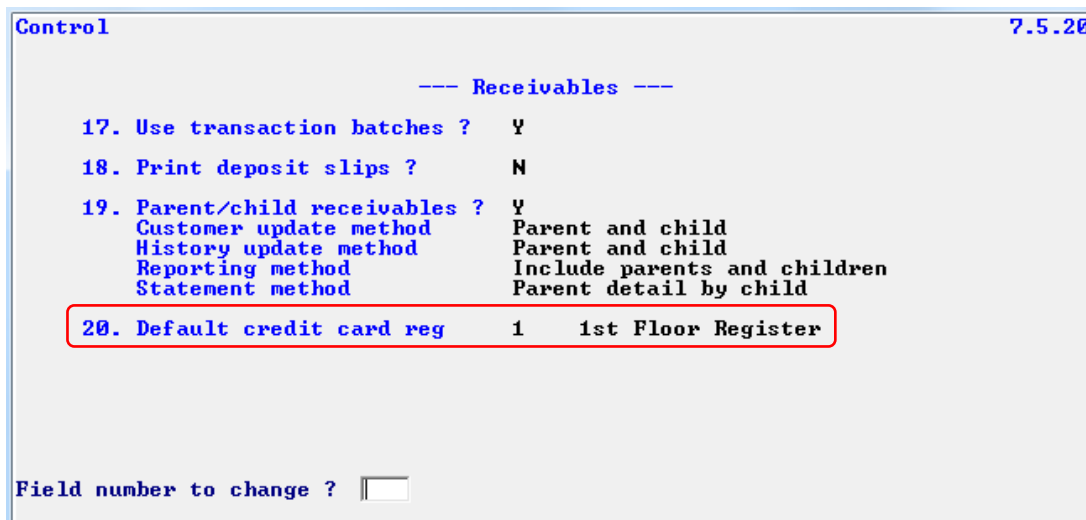
## Point of Sale Forms Printing Program

Of course adding all those fields to the POS forms designer doesn’t matter if we didn’t modify the POS forms printing program. So obviously we modified that quite a bit as well.

## Setup>Customers>Control

It is possible that CounterPoint is being used in a B2B environment with a single register or even no registers at all. In this situation the customers are likely set up to be “A/R” and they mail in checks, call in with credit cards or even have a tokenized card on file. As mentioned before, when one enters a program that needs to access an Ingenico CP-EMV prompts the operator for the register number. This allows the system to associate the Ingenico you will be using to your session. But what if the company only uses Cash Receipts to enter their receipts and they only use a single Ingenico to do so ? What if they don’t even use an Ingenico ? (More on that in a second.) Prompting for an Ingenico every time a user enters the Cash Receipts menu entry would be tiresome and needless.

To alleviate this we added a default register number field to the Customers control record. The field is *Setup>Customers>Control>“Receivables”>“20. Default credit card reg”*. Here’s what the screen looks like now.



```
Control 7.5.20
      --- Receivables ---
17. Use transaction batches ?   Y
18. Print deposit slips ?      N
19. Parent/child receivables ? Y
    Customer update method     Parent and child
    History update method      Parent and child
    Reporting method            Include parents and children
    Statement method           Parent detail by child
20. Default credit card reg     1   1st Floor Register

Field number to change ? 
```

The new field is used to control the register/store used when performing authorizations in *Customers>Cash receipts>Enter*. It allows entering a specific register value, pressing <F1> for “Not used” or <F2> for “Prompt”. If you enter a register number the cash receipts program will not prompt for a register number at startup and will use the Ingenico associated with that register for authorizations. The merchant credentials associated with this register’s store will be used for all authorizations. “Not used” means your A/R department doesn’t use an Ingenico at all. But that doesn’t mean you cannot authorize credit cards payments. You can still employ any tokenized credit cards your customer has on file. You can also manually enter the credit card number (and expiration date) when you key the transaction. But since CP-EMV doesn’t store credit card information in any files the transaction must be authorized immediately. The merchant credentials associated with the store number from the POS control file record will be used for all authorizations. “Prompt” means to ask for the register number every time you launch the cash receipts program. You would need to do this if your A/R department has multiple Ingenicos. The merchant credentials associated with this register’s store will be used for all authorizations. If you leave the register number blank (and skip the resulting warning) you will not be able to process any physical credit cards (because you haven’t tied your session to an Ingenico), but you will still be able to perform cardless authorizations. The merchant credentials associated with the store number from the POS control file record will be used for all authorizations.

Note that the Customers package is common to all stores in your enterprise. If you have more than one store each having its own A/R department you can't define a single register number here. Otherwise all of your stores' A/R departments would attempt to use the same Ingenico that's located only in one of the stores. They would also authorize credit cards to the same merchant account. Nor can you set the field to "None" since all cash receipt authorizations would use the merchant credentials from the store number in the POS control file.

CP-EMV allows automatically refreshing customer cards, but the customer cards option must be turned on. Merely turning on customer cards doesn't turn on automatically saving customer's tokens. But not having customer cards disabled prevents automatically saving all customers' tokens. Since customers are shared between all stores, individual stores cannot control the saving of customer cards (tokens). This is controlled via the Customer control file settings and (depending on the configuration) the customer's individual settings.

On the "Customer Cards" screen you'll find a new field "4. Auto-save customer cards". This field controls whether a token that is returned with successful authorizations is automatically saved with the customer's cards. It allows three values.

- "Always" Always save any token received as a customer card.
- "By customer" Save the token as a customer card if the new "Tokens?" field on the customer maintenance screen has been set to "Y".
- "Never" Don't save tokens for any customers.

It's pretty simple actually. See [Customers>Customers](#) for more information on the new "Tokens ?" field.

On the same screen is a table that defines the types of customer cards. There are eight "slots" available for entering customer card definitions. This is unmodified from vanilla CounterPoint. However we didn't like the way the cursor navigated once you were in the table making changes. So we added better navigation via the up/down keys.



## Setup>Order Entry>Control

As mentioned before, when one enters a program that needs to access an Ingenico CP-EMV prompts the operator for the register number. This allows the system to associate the Ingenico you will be using to your session. But what if the company only uses a single Ingenico ? What if they don't even use an Ingenico in Order Entry ? Prompting for an Ingenico every time a user enters the order maintenance program would be tiresome and needless.

To alleviate this we added a default register number field to the Order Entry control record. The field is *Setup>Order Entry>Control>"21. Default credit card reg"*. Here's what the screen looks like now.

```
Control 7.5.20

14. Print form after entry ?      N

15. Form ID <Orders>              INUP1 Invoice - Plain Paper single
16. Form ID <Invoices>            INUP1 Invoice - Plain Paper single
17. Form ID <Picking tickets>     PIKPP Picking ticket - Plain paper

18. Message lines on forms

19. Allow editing of notes in
    order view functions ?        N

20. Update ship date in post ?    Y
21. Default credit card reg       1    1st Floor Register

Field number to change ? ||
```

The new field allows entering a register value, pressing <F1> for "Not used" or <F2> for "Prompt". If you enter a register number the order maintenance program will use that for authorizations. So you won't need to key the register number every time you launch the order maintenance program. "Not used" means entering orders doesn't require an Ingenico at all. But that doesn't mean you cannot authorize credit cards payments. You can still employ any tokenized credit cards your customer has on file. "Prompt" means to ask for the register number every time you launch the order maintenance program. You would need to do this if you have multiple registers authorizing credit cards via Ingenicos.

Note that the Order Entry package is common to all stores in your enterprise. If you have more than one store each using Order Entry you can't define a single register number here. Otherwise all of your stores' would attempt to use the same Ingenico that's located only in one of the stores. They would also authorize credit cards to the same merchant account.

Because we added a numbered field to this screen it required us to bump the field numbers on the next screen.

## Point of Sale>Tickets>Enter

Visually not much has changed with the ticket entry program. Once you answer “Y” that the ticket is complete the message “Waiting for tender & authorization” is displayed. The message remains as long as the authorization process takes. Once the authorization completes (pass or fail) the message is replaced with another that displays the result of the authorization. If there is an error in the process an error message displays the nature of the error and the cursor returns to the pay code field. If a partial payment is authorized the remaining balance is calculated and displayed and the cursor returns to the pay code field.

If you are using customer cards the window that offers the card to use has changed. CounterPoint allows defining up to 8 cards for each customer. If you had defined cards for a customer prior to installing CP-EMV you might have actual customer card data (as opposed to tokens) on file. If you add card data after installing CP-EMV that card data was tokenized prior to writing the information to disk. So you could actually have a customer with both actual card data and tokens defined. In ticket entry, if the store being run has CP-EMV enabled the customer cards window only displays tokenized cards. If the store is still using CP-Gateway only non-tokenized cards are displayed.

When your registers lose their Internet connection the FCC Server (Figaro) will generate offline authorizations depending on the setting of your floor limit parameter. It is good to know when this is happening to allow you the earliest opportunity to correct the situation ASAP. To this end we’ve added a bit of code to display “\* Offline authorization \*” in the upper right corner of the regular ticket entry screen every time an offline authorization is received. This message will remain on the screen until the next successful authorization.

There is a possibility that an authorization is successful, but something happens in the communication back to CounterPoint. In this case CounterPoint will timeout and the clerk will assume he needs to attempt the authorization again. This is reasonable, but will also result in a double charge to the customer. To help alleviate this we have added code to the regular ticket entry program to look for consecutive authorizations for the same amount on the same masked credit card number. So if a register receives two authorizations for the same amount and the against the same masked credit card number the program will issue the message:



If a clerk sees this message they should probably do something. Typically a manager will log onto FreedomPay's Enterprise portal and void the duplicate charge. That's a bit inconvenient though since the manager will need to note the details of the ticket before walking to another machine to log into the portal. To simplify this we created a program ([System>View>EMV Transaction history](#)) that allows viewing EMV transactions and optionally (if you're a manager) voiding a transaction from within CounterPoint. From the ticket entry screen you can press <F10> to run the program via Quick Access. Locate the duplicated transaction, void it then return back to the ticket entry program and continue as if nothing had happened at all.

Note that the program doesn't check that the ticket number is different. The process may have got so boggled the clerk may have decided to rekey the ticket. But this would still result in a duplicate charge. So this check only looks at the amount requested and the masked card number it was charged against.

### **Point of Sale>End of Day>Post**

When POS tickets, orders and pay-on-accounts are posted the EMV transaction history records need to follow them from being associated with POS to Sales History. We modified this program to make that change.

We noticed that when the program was creating inventory cost distributions they were being loaded with the string "From PS Str#999 Reg#999". The problem is that string is 24 characters long and the distribution record's reference field (which stores the value) is only 20 characters long. So the last 4 characters were always being truncated. We removed the leading "From " and expanded "PS" to be "POS" resulting in a 20-character reference field formatted like this "POS Str#999 Reg#999".

### **Point of Sale>Offline operations>Create upload file**

### **Point of Sale>Offline operations>Import upload file**

We modified the Offline feature of CounterPoint to include the EMV transaction history file. There's not much to say here. We added a counter to the screen so the operator can see it happening. The EMV file goes in the expected directory with the expected name.

### **Point of Sale>Touchscreen**

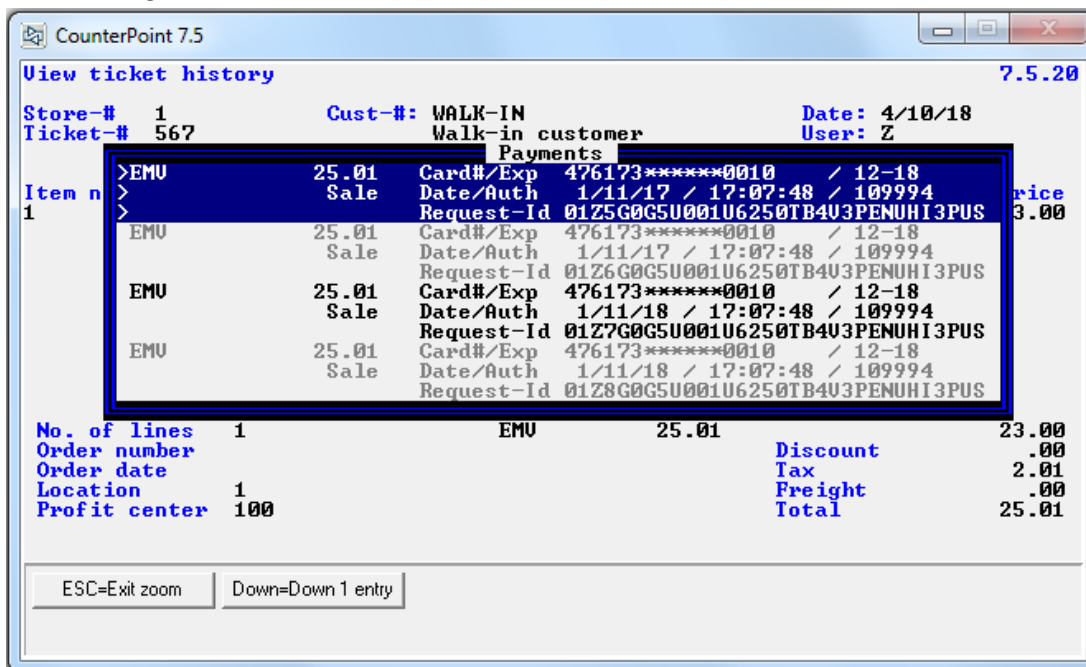
We added the ability to perform authorizations (including partial authorizations) to the POS Touchscreen program. However, since Synchronics never distributed the source code for the Touchscreen user interface we could not modify Touchscreen to support tokenized customer cards.

## Sales History>View>Ticket history

When the POS tickets and orders (or O/E orders) are posted they are written to the Ticket History files. They become visible in Sales History. We modified the View Ticket History program to employ the portable view EMV transaction history program. It replaced the original window that used to display the payment information. That program wasn't very aesthetic and didn't display much information if you needed to research a charge. Pressing <F8> while viewing a ticket will call the portable view EMV transaction history program to display the payment history of the ticket being viewed.

The new view EMV history program displays considerably more information which should help a great deal if you need to look the transaction up weeks or months later. The EMV information window displays the transaction details for all 6 tenders stored on the header record. (Usually there are only ever up to 3.) The window automatically resizes and repositions as required depending on the number and types of tenders displayed. Note that only the final tenders are displayed. If there were other EMV tenders associated with the ticket before it was posted (such as EMV voids), the information associated with these transactions are stored in the EMV Information file. You can display the complete EMV history for each ticket using the new [EMV Transaction history](#) report or [View EMV Transaction history](#).

The resulting window looks like this:



The vanilla program was quite limited in the tender history it would display. Also the window the history would display in was fixed in length regardless of the number of tenders being displayed. Our version of the payment window is much more sophisticated. The new window will size and position itself to the number of lines needed to display the payment history of the ticket. Instead of allocating a fixed 2 lines for each tender (like the vanilla CounterPoint) the new version displays one, two or three lines per payment transaction depending on the nature of the tender and the amount information available. So cash tenders display on a single line. Checks display on two lines and credit card payments will display on two or three lines. In order to easily discern each tender from the next the program displays them in alternating light and dark shades (like old-fashioned green bar paper). The “current” selected payment is displayed in dark blue. (The last payment transaction in the sample screen above.) You can navigate up and down through the payments using the up and down arrow keys. It’s simple and intuitive.

Unlike vanilla CounterPoint, CP-EMV remembers all of the credit card transaction history for each tender on each ticket. So if a credit card is (or several credit cards are) authorized, then voided then authorized again CP-EMV retains an EMV information record (and signature) for each transaction. This means you might have many more than a paltry three payment transactions to display. We allow up to 100 lines of payment information to be displayed. Of course any/all/none of these transactions may have a signature on file. If you are saving signature files locally the program checks to see if there’s a signature on file for each transaction. If there is the program displays an “S” on the right side of the entry’s second line. If that line is the selected line (being displayed in blue) the operator is offered <F1> to display the signature.

## **Customers Cards Maintenance Window**

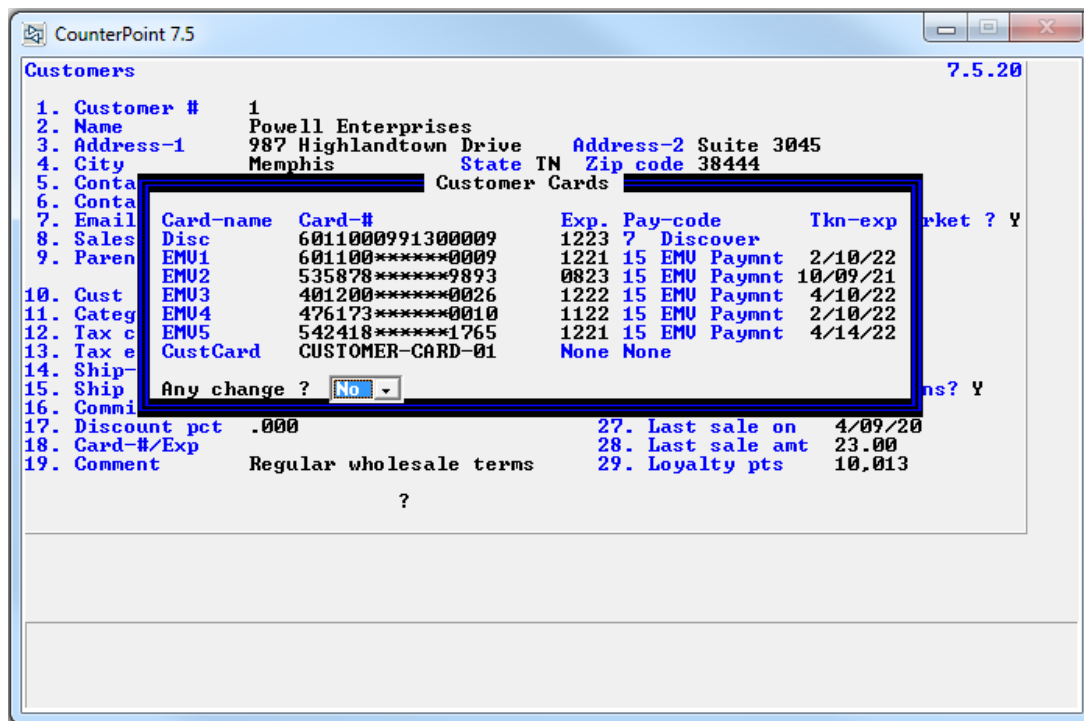
The program that allows maintenance to customer cards is its own program, but it doesn’t have its own full screen or even its own menu selection. The idea is this program doesn’t need a full screen and it might be useful to allow several programs to call it so you can maintain customer cards from several areas in CounterPoint. There are a number of such programs in CounterPoint. The most commonly known way to access this program is via the customer maintenance program by pressing <F7> at FNTC. But you can also call it from the regular ticket entry program by pressing <F4> while in the customer “zoom” screen. And there are several other areas in CounterPoint that we’ve added this ability.

This program allows maintaining up to 8 cards for a customer. Before you can do anything with customer cards you need to turn Customer Cards on ([Setup>Customers>Control>“Customer Cards”](#)). See CounterPoint’s documentation for details on that. Customer cards allow you to bill the customer without their card being present. That’s probably not so important in a retail environment with a lot of “walk-in” customers, but it can be very important in a B2B or A/R-based store. You might have “live” credit card information on file for many customers so you can charge them without their credit card being present. You don’t have to give up that convenience when using CP-EMV. However you will need to convert the credit card information to tokens to be out of PCI scope. We’ve already discussed what tokens are in the “Tokens” section so we won’t repeat all that here.

When the program first starts it will prompt you for “Credit card register”. That’s because you might want to swipe the customer’s credit card in order to create a token. CP-EMV needs to be able to associate the running program to the right Ingenico to perform the swipe. When you call the program from ticket entry the program doesn’t prompt for the register number because CounterPoint already knows what register you’re at. It just passes that register number to the programs as it calls it. Enter the register number associated with the Ingenico you’ll be working with. If you don’t enter a register you won’t be able to swipe a card to create a token (but you will be able to manually enter and delete entries).

Once you’ve entered the register number you’ll be presented the same credit card maintenance window as in vanilla CounterPoint with a slight difference. Just to the right of the pay code fields you might see a value for the token expiration date field. That indicates the credit card information being displayed on that line is actually derived from a token – not actual credit card information. The credit card number will be masked. Ideally all of the entries will have a masked credit card number and a token expiration date. Otherwise you’re still in PCI-scope.

Here’s what that screen looks like.



At this point I have to warn you that this program has a psychotic manner about it. It’s OK though – it’s “by design”. We designed CP-EMV to allow mixing stores using CP-Gateway with store using FreedomPay. So one CounterPoint installation – using the same customer file – can store credit card information in the old CounterPoint manner (a liability) AND store tokens for those stores using FreedomPay (the smart stores). So this program needs to support both credit card information for CP-Gateway functions and create tokens for FreedomPay. Thus the psychotic behavior.

But supporting doesn't mean endorsing. The idea is to stop using CP-Gateway style credit cards. You can keep the old entries in there and CP-Gateway stores will be able to use them, but any new entries should be tokenized for use with FreedomPay. The software looks at how the pay code has been set up to know whether to tokenize a card or not. If the pay code is a credit/debit card, but it doesn't have any validation fields (typically you would prompt for the card number and expiration date) the program assumes the pay code is an EMV card and thus should be tokenized. Ideally all the stores will immediately adopt FreedomPay processing. That's what you paid for right ? Eventually the old CP-Gateway cards will expire and their liability will with it – as well as the store's ability to use those old (shockingly unencrypted) credit cards. Or even better, you can tokenize all those unencrypted credit cards (making them PCI-compliant) then purge them from your system entirely. You read about that [HERE](#).

Entering entries works exactly as before, but even better. You key the credit card number and the program performs a check to see if the value entered makes any sense at all. Then you enter the expiration date. The program then contacts FreedomPay to generate a token. If that works the program displays the token expiration date and the cursor continues to the next entry (or "Any change ?"). The tokenizing process is very fast. It's faster than obtaining an authorization. Note that this process works even if you don't enter a register number when the program first starts. If you key the information the program doesn't need access to an Ingenico so no associating register required !

Here's the "but better" part. If you entered a register number prior to opening this window you will have the ability to press <F1> to swipe a card at any entry. That's a feature of CP-EMV. This will wake up the associated Ingenico and prompt for the card to be swiped. Swipe the card and all of the fields will automatically display (including the token expiration date) and the cursor will continue to the next entry (or "Any change ?"). Note that you don't have to clear a populated entry prior to reloading it with a token. Just press <F1> at the entry and the program will overwrite the existing information. Also note that the Ingenico will prompt you to swipe the card – not insert the card. For some reason when Susanna prompts an Ingenico to collect the card information (as opposed to tendering a payment) it wants the card to be swiped only. Weird.

If there's an error you'll receive an error message, but if you entered the correct register number and the Ingenico is alive and functioning it will most likely work.

We also added the ability to press <F3> to clear entries. Previously you would have to delete all of the digits from the field to indicate that you wanted to clear the entry.

## Customers>Customers

A feature of CP-EMV is to automatically save customers' tokenized credit cards as customer cards. See the section "[Tokenized Customer Cards](#)" for more information on this. To control the updating of individual customers' cards we've added a new field simply named "Tokens?" to the customer maintenance screen. This screen was already pretty full so we had to add the field as part of the existing "Allow checks?" field. Here's what the screen looks like now.

```
Customers 7.5.20
1. Customer # 1
2. Name Powell Enterprises
3. Address-1 987 Highlandtown Drive Address-2 Suite 3045
4. City Memphis State TN Zip code 38444
5. Contact-1 John Powell Phone-1 901-654-1234
6. Contact-2 Phone-2
7. Email turnerANDhooch@upsp.com Market ? Y
8. Sales rep 789 Avram Goldberger
9. Parent cust

10. Cust type Cash 20. Ecommerce ? N
11. Category WSL WSL - Description 21. Unposted bal 51.02-
12. Tax code MEM Tennessee state tax 22. Credit limit 1.000
13. Tax exempt # 23. Credit rating A
14. Ship-via T Truck 24. Credit hold None
15. Ship zone 25. Allow checks? Y Tokens? Y
16. Commis pct <Not applicable> 26. First sale on 3/31/04
17. Discount pct .000 27. Last sale on 9/23/20
18. Card-#/Exp 28. Last sale amt 23.00
19. Comment Regular wholesale terms 29. Loyalty pts 12.018

Field number to change ? 
```

This new field only matters if you've set auto-saving tokens to "By customer". If you've set auto-saving tokens to "Always" or "Never" this field displays "n/a" in light grey. It's pretty simple. If you want the customer to automatically keep their customer cards updated set the field to "Y".

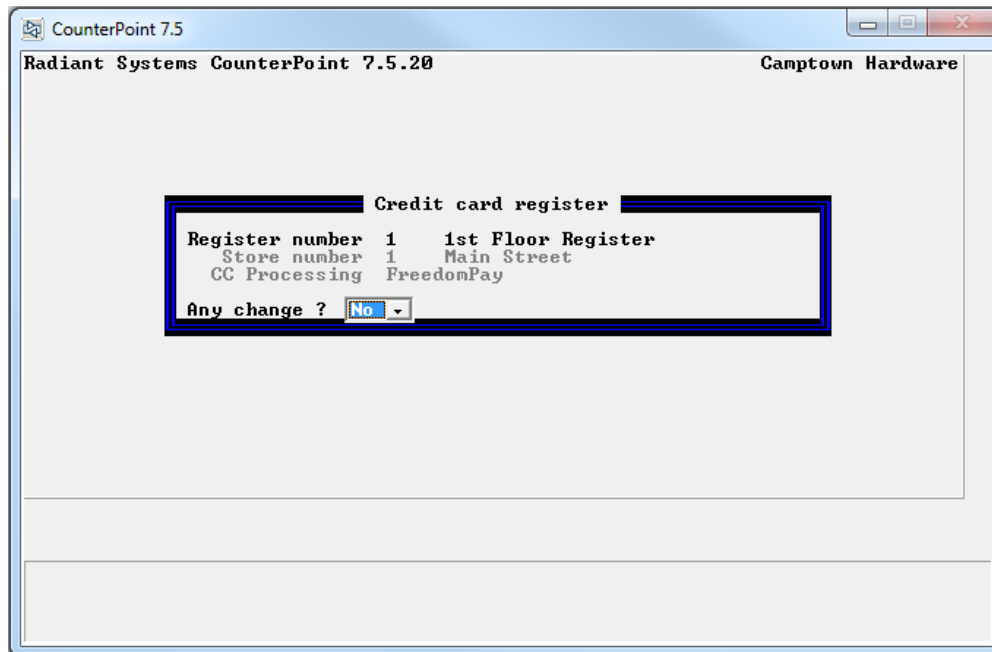
Keep in mind that this field doesn't control whether a customer can have tokenized customer cards. All customers can have them. It only controls whether they are automatically updated.



## Customers>Cash Receipts>Enter

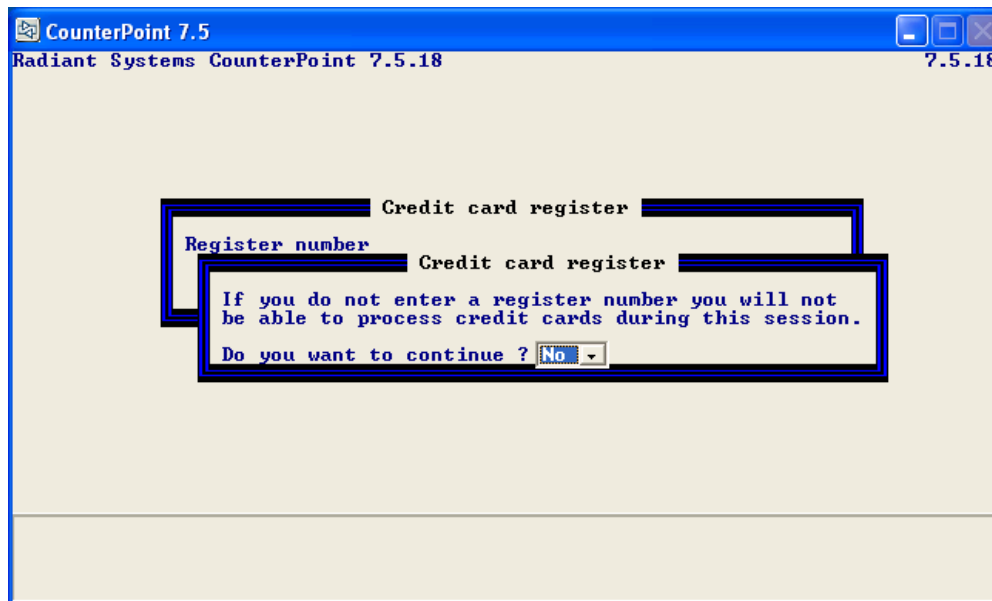
We added the ability to perform EMV payments to Cash receipts as well. The process works pretty much the same as before, but there are a few enhancements.

Vanilla CounterPoint communicates directly with the credit card terminal. It controls the terminal and sees all the raw, unencrypted information on the credit card. That's one of the liabilities that CP-EMV eliminates. CP-EMV employs FreedomPay's Susanna service to communicate with the Ingenico. Since Susanna uses a workstation-id to map the currently running instance of CounterPoint to a particular Ingenico, if you intend to utilize an Ingenico to accept credit cards in cash receipts you need to specify the workstation-id you'll be using to swipe/dip the credit cards. The value you've assign to the field *Setup>Customers>Control>"Receivables">"20. Default credit card reg"* controls whether you are prompted for a register number. If you provided a register number in this field the program will use that register number and will not prompt for a register number. Likewise, if you set the field to "Not used" the program understands A/R doesn't use an Ingenico and so won't prompt for a register number. Otherwise the program prompts the operator for the register to be used during the session. Here is a sample screen of that.



The operator enters the register number associated with the Ingenico he intends to use during the session. The program then displays the store info the register is associated with and a literal indicating whether the store is using CP-Gateway or FreedomPay for credit card processing. The register is then locked from being used by another workstation in the same way Point of Sale does. If another workstation attempts to open the same register, that operator will receive the ubiquitous "Register in use at another station" message. That prevents two workstations trying to send authorization requests to the same Ingenico. The same check is made if you have assigned a register number in *Setup>Customers>Control>"Receivables">"20. Default credit card reg"*. If the register is locked you'll receive the message "Default register in use at another station" and the program will prompt for a different register number.

If the operator doesn't enter a register number the program provides the following warning:



If the operator answers "N" the cursor returns to the register number field. If the operator answers "Y" the program continues to the entry screen in the normal manner. The wording of this second screen is a bit misleading, but we couldn't explain everything on a couple of lines. Basically, if you answer "N" you can still enter cash receipts and even apply credit card pay codes to them. You can even get authorizations for tokenized cards and hand-keyed credit card numbers. We merely left the message with the same verbiage as in the other areas of CounterPoint that prompts for the register number.

Note that the value keyed for the register determines the physical Ingenico to be used. The register number and the associated store number are passed along with the authorization request to allow Figaro to determine which Ingenico to address for the authorization. The merchant-id the card is processed under is still the current store number as defined in *Setup>Point of Sale>Control>"1. Store number for this location"*. In fact, all default values come from this store record just like in vanilla CounterPoint cash receipts program. The register number entered ONLY determines the Ingenico being addressed. For this reason the register number entered must be associated with the current store number.

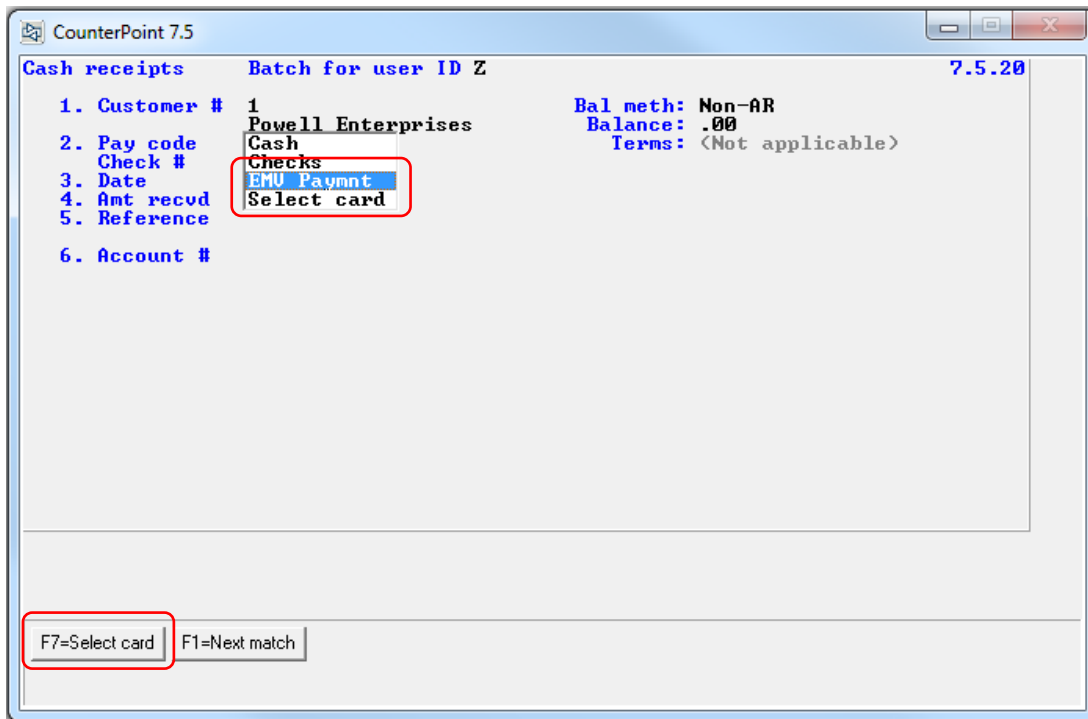
If you need to perform authorizations towards different stores' merchant credentials you'll need to enter a register specific to the desired store while performing authorizations for that store. To perform authorizations for a different store you need to exit the cash receipts entry program and come back in to enter a register number associated with the new store number.

If you use an Ingenico for cash receipts you will probably want to define a back-office "register" to enter/authorize A/R payments. Obviously you probably won't have the customer there to sign the Ingenico so you should connect a non-signature capture Ingenico (such as a 320 or 350) to these back office registers.

Vanilla CounterPoint allowed entering the credit card information, and authorizing the transaction immediately or in a batch later or during posting. By their nature, EMV cards must be processed “up front” because the system doesn’t store the credit card information. So the authorization must take place in the transaction entry program. However CP-EMV does allow cash receipt transactions to be keyed and a credit card pay code assigned (or a customer’s tokenized card assigned), but not authorized immediately.

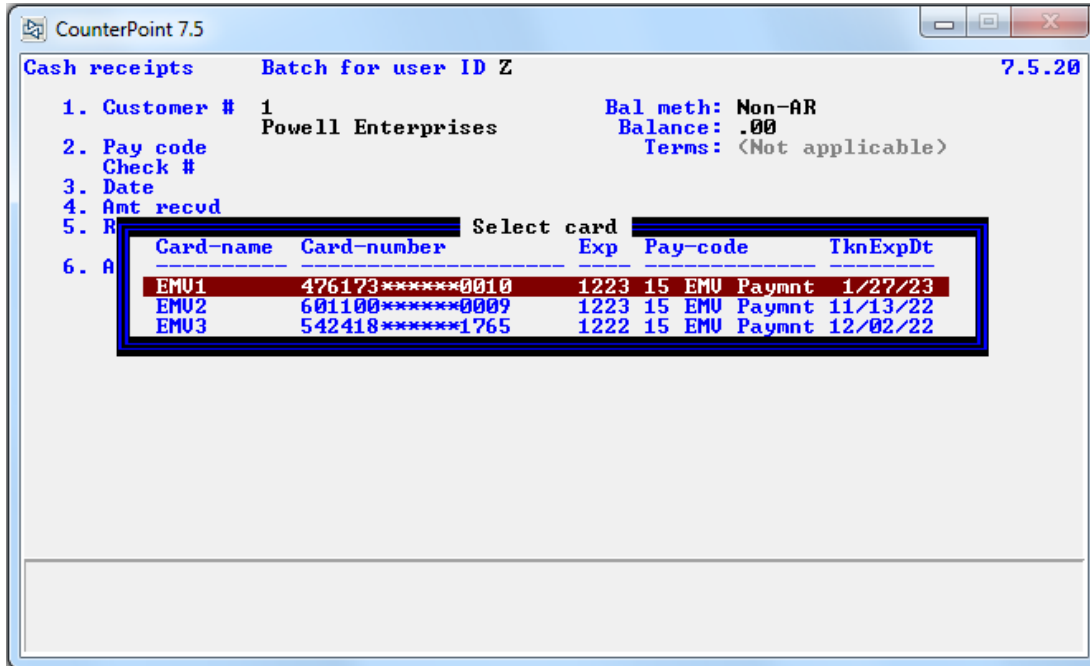
CP-EMV allows using tokens in lieu of a customer’s credit card. Vanilla CounterPoint allowed storing customer credit card information for up to 9 credit cards. Of course that’s exactly the liability that CP-EMV is tasked to eliminate. That leaves a bit of a hole when it comes time to authorize a payment for a A/R customer. You don’t have the card on hand and it’s not PCI-compliant to have that sort of information written down for keying into the Ingenico. CP-EMV allows storing customer card tokens in lieu of the actual card data. CP-EMV’s version of cash receipts allows you to use a customer’s token as payment just like a credit card. The token functionally represents the credit card.

If the customer has tokenized customer cards defined (See [Customers>Customers](#) above) the pay code field will offer cash, check, EMV credit cards and a new selection “Cust cards”. It will also offer <F7> for “Cust cards”.



If the customer only has a single tokenized customer card on file the <F7> prompt displays “Card on file” instead of “Select card”. That’s because there’s only one card on file right ? Why bother open the card selection window if there’s only a single card on file ? So pressing <F7> in this case immediately assigns the customer’s card to the cash receipt without opening the card selection window.

But if the customer does have more than one tokenized customer card on file selecting “Select card” (or pressing <F7>) will open a popup window displaying the customer’s tokenized credit cards.



Use the up/down keys to highlight the desired card and press the <ENTER> key. The pay code associated with the token and the card number will be displayed and the cursor will proceed to the date field. Entering the rest of the fields is vanilla CounterPoint. As previously mentioned, when you get to FNTPC the program will allow you to authorize the card immediately or you can save the transaction as-is and get the authorization later.

If you enter a new cash receipt to be paid via an EMV credit card pay code you have several methods to enter and authorize the customer’s credit card information. At the field “2. Pay code” you can select the pay code you have defined for EMV credit cards. You can also select “Select card” from the same list or press <F7> (functionally the same thing) to open a window on the screen to select one of the customer’s previously tokenized credit cards.

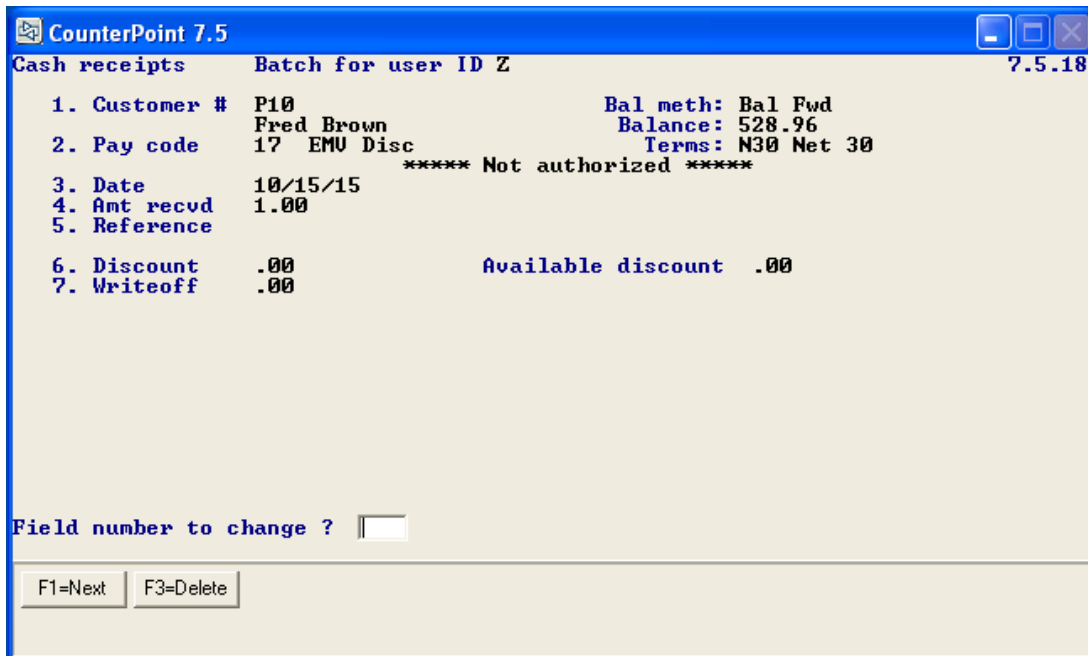
At this point the importance of the field *Setup>Customers>Control>“Receivables”>“20. Default credit card reg”* becomes really important. How the program operates from this point is dependent on how that field is set. So let examine each situation.

First let's assume that *Setup>Customers>Control>"Receivables">"20. Default credit card reg"* has been assigned a register number. So when you launched cash receipts the program did not prompt you for a register number. It got it from the control record instead. It also tells the program that you have an Ingenico attached. Since you have an Ingenico attached the program will expect you to use it to swipe the customer's credit card or manually enter the customer's card information. Of course you can still select one of the customer's tokenized cards for payment. At FNTC the program offers a couple of new function keys. "F7=Cust cards" allows opening the customer card maintenance window. This would allow you to tokenize a credit card for the customer prior to using it as a tender. "F8=Authorize" wakes up the Ingenico to swipe the customer's card for the authorization.

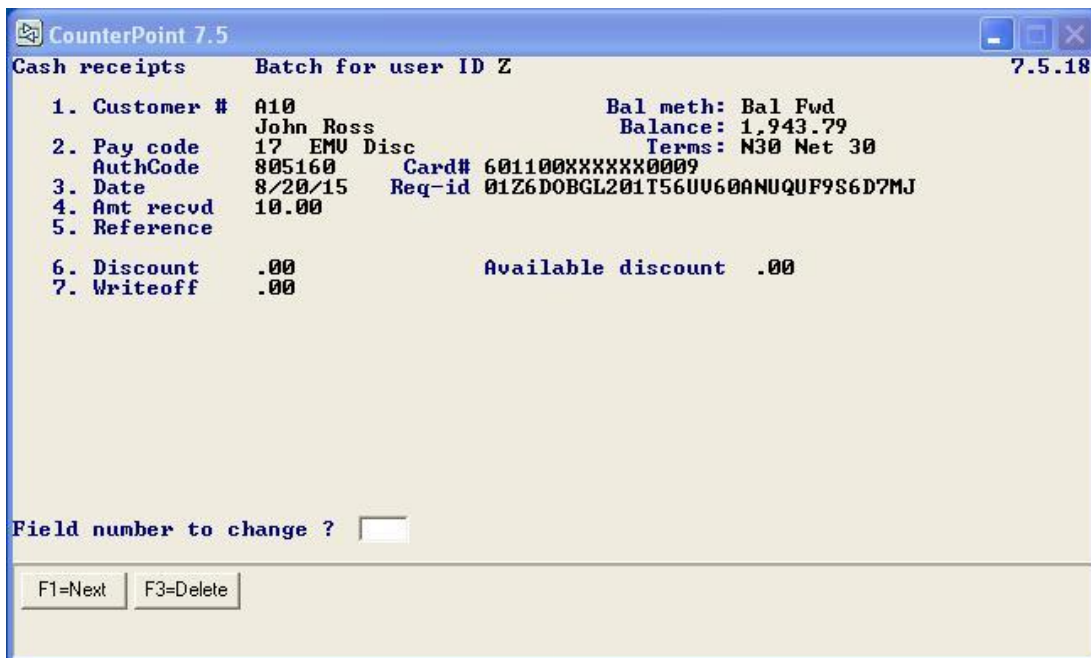
Now let's assume that *Setup>Customers>Control>"Receivables">"20. Default credit card reg"* has been set to "Not used". In this scenario the program knows it is not working with an Ingenico at all. So it doesn't prompt for a register number. Since there's no Ingenico, swiping a credit card or manually keying the card information on the Ingenico is not an option. In this mode the program will allow you to select a tokenized credit card or manually enter the customer's card information on the screen. If you entered the transaction without selecting a tokenized card (so it has an EMV pay code but no card information) and the cursor is at FNTC the program offers "F7=Cust cards" and "F8=Keyed-CC". Pressing <F7> opens the customer card maintenance window. This would allow you to tokenize a credit card for the customer prior to using it as a tender. "F8= Keyed-CC" prompts for the credit card's number and expiration date fields. Once those fields are entered and the cursor has returned to FNTC the program changes the <F8> prompt to "Authorize".

The last setting for *Setup>Customers>Control>"Receivables">"20. Default credit card reg"* is "Prompt". As the literal implies the program prompts for the register number. If one is not provided the program basically falls back into "Not used" mode described above. Otherwise it acts as in the first mode where a default register number was provided in the Customer control record. Note that if you press <F7> for "Cust cards" at FNTC the program will again prompt you for a register number. That is because the customer card maintenance program assumes it will use an Ingenico to swipe the customer card to tokenize. But you can leave the register number field blank, answer "Y" to the confirmation prompt and key the card information manually to create tokens.

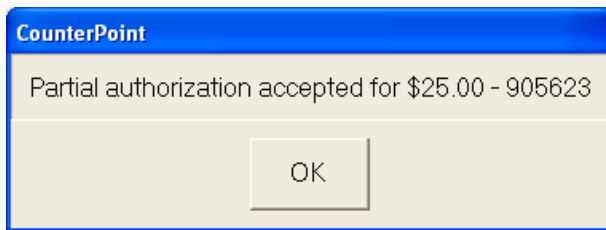
In all cases, if the transaction has what it needs to attempt to get an authorization, pressing <ENTER> at FNTC will result in the program asking "Do you want to authorize now?". Answering "Y" wakes up the Ingenico to swipe/dip/key the credit card information and starts the authorization process. Or the operator can answer "N" and authorize the transaction later. When viewing previously keyed, but unauthorized transactions the program offers <F8> to authorize the current cash receipt. If an EMV pay code is applied and the authorization isn't performed the literal "\*\*\*\*\* Not authorized \*\*\*\*\*" is displayed as a reminder as the example below depicts.



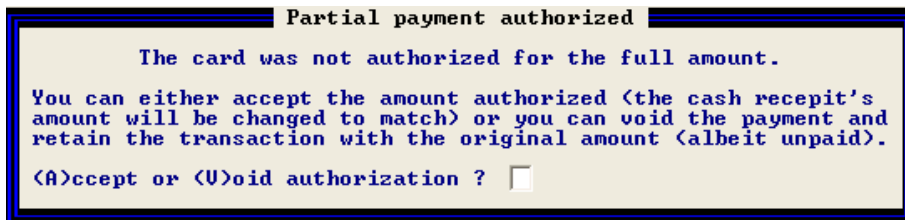
When you authorize the charge you get one of three responses. If there's an error you'll receive a message displaying the error number and a (very) brief description of the error. Figure it out and try again or save the transaction and try again later. If the authorization was successful you will/won't receive a message depending on the field *Setup>Point of Sale>Stores>Stores>Configuration options>screen#-7>"10. Suppress messages/dialogs for: Successful card/check authorization"*. An authorized transaction displays the authorization code, the masked card number and the EMV transaction's request-id.



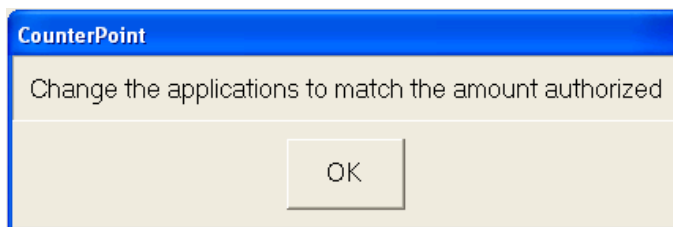
The third response is rare, but possible – a partial payment. This will occur if the card tendered is a debit card that doesn't have the monies to cover the total amount or the credit card exceeds the account's credit limit. If the authorization returns a partial payment an appropriate message displays notifying the operator of the partial payment (playing the partial payment sound if it's set up) and displaying the authorized amount and the auth code.



Then the following message is displayed.

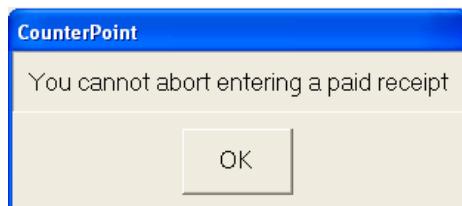


If the operator answers "Void" the program will void the authorization and the cursor will return to FNTC. If the operator answers "Accept" the cash receipt's amount field is automatically changed to the amount authorized. If the customer is an open item customer the amount(s) applied no longer match the amount received so the program displays another message.



At this point the operator needs to modify the apply-to fields as required to match the amount that was actually authorized. The message will redisplay every time the operator presses <ENTER> at FNTC until the apply-tos match the amount authorized.

Note that attempting to abort entering a paid (via EMV payment) will result in the message:



Thus an authorized transaction must be completely entered. If the operator wants to cancel the transaction after it has been authorized he will need to complete the transaction, pull it back up in change mode and delete it there. When an EMV-tendered cash receipt is deleted the payment is automatically voided.

So what happens when you delete a cash receipt that has already been authorized ? CP-EMV first attempts to void the authorization. If the void is successful you'll get the message "Payment voided successfully" and the cash receipt will be deleted. If too much time has passed (sometimes it's not much time at all) the authorization cannot be voided. In this case CP-EMV will attempt to refund the amount using the token that was created during the transaction's prior authorization. If that is successful you'll get the message "Return authorization successful". That will likely work. But if it doesn't you'll receive the message/prompt "Cannot void prior auth. Do you have card info to create refund trx ?". CP-EMV is offering you one last opportunity to reverse the charge by using either the physical card or keying in the customer's card information. If you have the card (or the card's information) answer "Yes" and CP-EMV will create a refund authorization for the amount. Otherwise you'll get the message "Return authorization for \$99,999.99 failed - Trx not deleted" and the cash receipt will not be deleted.

Unauthorized transactions are skipped over in the posting program until they are authorized. In the Edit List, unauthorized transactions display the literal "\*\*\*\* Credit card needs to be processed \*\*\*\*". Authorized transactions update the new EMV information file.

If a signature was collected when the cash receipt was authorized, CP-EMV sees there is a signature on file for the transaction and offers <F5> for "SigCap". Pressing <F5> will open a window displaying the saved signature file. We have a whole section on configuring and displaying signature captures. You can find that discussing [HERE](#).

If the program finds an EMV transaction history record for the transaction it offers <F4> for "EMV Info" to view them. Pressing <F4> calls the portable EMV transaction viewing program which displays all of the EMV history for the transaction in "greenbar" form. You can details on that feature [HERE](#).

BTW – We also added the ability to <CTRL+F1> to go backwards through the transactions on file. We also removed the <F7> "Voice auth" option. We don't know anyone that does this anymore, FreedomPay doesn't support it and we needed the function key for something more useful.

When a cash receipt is posted the transaction creates an A/R open item. During the posting process any EMV transaction records associated with the transaction is "moved" to follow the cash receipt to the open item record. But what if the cash receipt is never posted ? What if the cash receipt is keyed, authorized, the authorization is voided then the cash receipt deleted ? The EMV transaction history records never migrate to open items. Programmatically we could delete the EMV transactions, but then we're losing history for at least two actually EMV transaction events. So we leave the EMV history where it is – associated with cash receipts. This way they still show up in EMV transaction history reports and viewing programs. This might be confusing if one were to run such a report and see EMV entries for cash receipts that don't exist. That's why they're there. If you find them offensive you can always purge them using [System>Purge>EMV Transaction history](#).



## **Customers>Cash Receipts>Edit list**

## **Customers>Cash Receipts>Post**

The program that produces the cash receipts edit list is the same program that posts cash receipts. So we've lumped them together here because they share the same changes to the resulting report.

We added an additional detail line to the report that displays the status of an EMV tender (if there is one). If the tender still needs to be authorized the program displays "\*\*\* Credit card needs to be processed \*\*\*". If it has been authorized it displays a few details about the authorization from the EMV transaction history file. We also removed the asterisk character that prints next to the auth code to indicate a voice authorization (which doesn't happen anymore). That allowed us to remove the second report legend line that remarked on that. You can see an example of the resulting report [HERE](#).

The program also moves the EMV records that are associated with a cash receipt transaction to be associated with the generated open item record.

## **Customers>Month-end>Close open items**

We modified this program to move the EMV records associated with any open item records that are closed to their matching closed item record. If the open items are purged the associated EMV records are deleted instead.

## **Customers>Change open items>Change open items**

This program allows changing fields that comprise the primary key for the open item records. The record's primary key is what associates an open item record with its matching MEV records. So we modified this program so that when an open item's primary key is changed, its associated EMV records are also updated.

## **Customers>View>Open items**

## **Customers>View>Closed items**

When a credit card cash receipt is posted the posting program associates the payment-type open item with the EMV transaction information. The vanilla version of the View Open Items program doesn't display credit card information. We modified these programs to check if the highlighted entry has any EMV transaction history records associated with it. If so, the program offers <F8> to view the EMV payment information. The resulting window works exactly the same way as described previously. (In fact, the same program displays the EMV information system-wide.)

## Customers>Reports>Customer cards

This new program allows listing your customers' customer cards using various useful parameters. You can use it to determine the health and "staleness" of your customer card data. It can also provide you information on how and when to flip your older CP-Gateway customer cards over to tokens. It's pretty fancy because it has the option to employ one of two different parameter screens – the "by parameter" screen and the "by position" screen. The "by parameter" screen filters the records based on the typical filtering values you've seen in many CounterPoint reports. The "by position" parameter screen filters records based on the position of the customer card in the customer card definition table as it appears in the *Setup>Customers>Control>"Customer cards"*.

Here's the "by parameter" parameter screen.

```
Customer cards list 7.5.20

Please enter:

1. Customer-# range      "First"      to "Last"
2. Customer on file ?   "All"
3. Cards to include     Both
4. Card exp date range  "Earliest" to "Latest"
5. Token exp date range "Earliest" to "Latest"
6. Include suspicious ? Yes      Export ? Yes

Field number to change ? 
```

- "1. Customer-# range" This parameter filters the records by customer number. It supports <F1> for "First" to "Last", <F2> to switch to the "positional" parameter screen and <F9> for DataLookup.
- "2. Customer on file ?" This parameter might seem a bit weird at first. It's asking whether to include customer cards if the customer record is on file or not. It's intended to allow deleting orphaned customer card records. It allows "Yes", "No" or "All". If you don't have any orphaned customer card records, setting this field to "No" will result in the message "No records in range selected".
- "3. Cards to include" This parameter offers 5 selections – "CP-EMV", "Gateway", "Both", "Neither" or "All". When filtering on this parameter the program first must determine if the card is a CP-EMV card, a CP-Gateway card or neither. If the card record has a non-blank value in the token field it is assumed to be a tokenized CP-EMV customer card. Failing that if the card has an expiration date it is assumed to be a CP-Gateway card. Otherwise it is assumed to be neither.

“4. Card exp date range” This parameter filters the records by the expiration date of the credit card – not the token’s expiration date. It supports <F1> for “Earliest” to “Latest”. You may have unexpired tokens for credit cards that have already expired. Why waste the disk space and possibly prevent the customer from having a newer card added automatically ?

“5. Token exp date range” This parameter filters records on the token expiration date. If the parameter “3. Cards to include” indicates not to include CP-EMV cards this parameter has no purpose and is disabled. It supports <F1> for “Earliest” to “Latest”. You could have unexpired credit cards on file with expired tokens. Again, why waste the disk space and possibly prevent the customer from having a newer card added automatically ?

“6. Include suspicious ?” Over time and versions the customer cards records can get pretty hosed. Despite their small size it is amazing the number of ways the customer cards records can get trashed.

This parameter allows three values – “No”, “Only” and “All”. Remember that this parameter is asking whether to include suspicious records so answering “No” excludes them. “Only” creates a report that contains only suspicious records (subject to the other parameters as well). “All” includes “clean” and suspicious records.

If you select “Only” the program assumes that you are attempting to identify and clean your customer card records. To help facilitate this, the program offers to export these suspicious records in the same directory and format as the Customer file utilities. Once the records have been exported you have the ability to manipulate them directly (via your favorite text editor) and import them back into CounterPoint.

There’s a bit more on this subject in [Customers>Purge>Customer cards](#).

Note that many of the conditions that can label a record as suspicious depends on the assumption that the card is either a CP-Gateway card or a CP-EMV card. This is a hard thing to discern programmatically. We settled on the following assumptions. If a record has a non-blank token field it must be a CP-EMV card. If the token field is blank and the record has a card expiration date, it must be a CP-Gateway card.

This parameter attempts to identify customer cards records that have something wrong with them. There are many conditions the program looks for, but there are probably some we didn’t think of. So don’t assume this program will find and report them all. Below is a list of the conditions it can identify. Each is labeled with the alphabetic error code that is referred to in the report.

- A. Each record is indexed into the customer cards definition table. This table allows up to eight entries. So if a card has an index of zero or greater than eight it is invalid.
- B. The record's index references a blank entry in the customer cards definition table.
- C. The record's index has been duplicated by another record for that customer.
- D. The record is a CP-EMV card, but the customer cards option isn't turned on or saving customer credit cards has been turned off or the customer has saving customer cards turned off.
- E. The record is a CP-Gateway card, but the customer cards option isn't turned on.
- F. The record is a CP-EMV or CP-Gateway card, but the card number isn't 15 or 16 digits in length.
- G. The record is a CP-EMV or CP-Gateway card, but the pay code in the customer card definition table is zeros.
- H. The record doesn't have an expiration date, but the entry in the customer card definition table has expiration dates enabled.
- I. The record is a CP-EMV or CP-Gateway card, but the pay code index assigned in the customer card definition table references a blank pay code in the store record.
- J. The record is a CP-EMV or CP-Gateway card, but the pay code index assigned in the customer card definition table references a pay code in the store record that is not a credit or debit card type.
- K. The record is a CP-EMV or CP-Gateway card whose expiration date is invalid.
- L. The record is a CP-EMV card whose token contains invalid characters.
- M. The record is a CP-EMV card whose token expiration date is invalid.
- N. The record is a CP-EMV card, but the pay code index assigned in the customer card definition table references a pay code in the store record that has non-blank reference fields. CP-EMV pay codes are set up without these reference fields.
- O. The record is a CP-Gateway card, it has a non-blank value for the token field and/or a non-blank (or non-zero) value in the token expiration date field.
- P. The record is a CP-Gateway card, but the pay code index assigned in the customer card definition table references a pay code in the store record that doesn't have reference fields defined. CP-Gateway pay codes are set up with these reference fields (i.e. "Card number" and "Expiration date").

Pressing <F2> at the customer-# parameter of the “by parameter” screen switches the screen to the “positional” parameters screen. This parameter screen filters the records based on their position in the customer cards definition table as it appears in the *Setup>Customers>Control>“Customer cards”*. This allows you to list the cards that have been assigned by their “slot” in the table. It’s probably less useful to you as a listing. But this program is also purges customer cards. Filtering records by their position is very useful for that purpose because it allows you to purge all of your CP-Gateway customer cards in a single pass. You can access that functionality via [Customers>Purge>Customer cards](#).

Here’s the “by position” parameter screen.

```

Customer cards list 7.5.20

Please enter:

1. Customer-# range      "First"      to "Last"

2. Customer card types  Card-name  Exp  Pay-code  Include ?
Discover                Y    7  Discover   Y
EMU1                    Y   15  EMU Paymnt N
EMU2                    Y   15  EMU Paymnt N
EMU3                    Y   15  EMU Paymnt N
EMU4                    Y   15  EMU Paymnt N
EMU5                    Y   15  EMU Paymnt N
EMU6                    Y   15  EMU Paymnt N
CustCard                N

3. Include suspicious ? Yes      Export ? Yes

Field number to change ? 

```

- “1. Customer range”      This parameter filters the records by customer number. It supports <F1> for “First” to “Last”, <F2> to switch back to the regular parameter screen and <F9> for DataLookup.
- “2. Customer card types”      This parameter basically just lists all of the entries in your customer card definition table in *Setup>Customers>Control>“Customer cards”* and allows you to answer “Yes” or “No” as to whether you want them to appear in the listing.
- “3. Include suspicious ?”      Over time and versions the customer cards records can get pretty hosed. Despite their small size it is amazing the number of ways the customer cards records can get trashed. This parameter attempts to identify customer cards records that have something wrong with them. There are many conditions the program looks for, but there are probably some we didn’t think of. So don’t assume this program will find and report them all. There’s a lot to say about this parameter. Since it has already been discussed (at length) in the section above (discussing the “by parameter” parameters) I’ll refer you there if you want more information on this parameter.

If the user indicates to create the export files both files are created in the EXPORT directory used by the regular file utilities programs. The “export” file is named and formatted the same as the regular file utilities would create (PSTRKF.EXP). The CSV file format is named PSTRKF.CSV. The first record of the CSV format file is a header record describing the column names. Note that customer card records are not the only type of records that are stored in the POS Tracking file (PSTRKF). Plus only suspicious records are exported. So if you choose the correct and import suspicious records back into CounterPoint you’ll need to delete the original suspicious records from the production file first then ADD the corrected records or specify to REPLACE the records in the existing file. If you “Create a new file” you’ll lose all the other records in the tracking file.

The report’s format is the same regardless of the parameter screen used. See the end of this document for an [example](#) of the resulting report.

### **Customers>Reports>Open item detail**

We added two new EMV fields to the open and closed items record layouts. So we added them to this report as well. When the report is run in “Full” format there is a additional detail line printed for those records with an associated EMV record. For a sample of the resulting report see [HERE](#).

### **Customers>Purge>Closed items**

This existing program allows purging closed open items within CounterPoint. It is quite possible that the records being purged have EMV transaction history records associated with them. If you’re purging the closed items you probably don’t need their associated EMV records anymore either.

So we modified this program to delete the EMV transaction history records that are associated with each closed item being purged. There is no indication this is happening. There are no changes to the parameters screen the resulting report.

## Customers>Purge>Customer cards

This is the same program as the similarly named listing ([Customers>Reports>Customer cards](#)), but this version unlocks the ability to delete the records it reports. If you merely want to do some reporting use the listing program. Don't risk accidentally purging your data. Since this menu selection can quickly purge all of your customer cards you might want to control who you allow to run it. Being the same program as the listing program it has the same parameter screens with the exception of the added "Print or purge ?" parameter. The "by parameter" parameter screen filters the records based on the typical filtering values you've seen in many CounterPoint reports. The "by position" parameter screen filters records based on the position of the customer card in the customer card definition table as it appears in the *Setup>Customers>Control>"Customer cards"*.

Here's the "by parameter" parameter screen.

```
Purge customer cards 7.5.20

Please enter:

1. Customer-# range      "First"      to "Last"
2. Customer on file ?   "All"
3. Cards to include     "All"
4. Card exp date range  "Earliest" to "Latest"
5. Token exp date range "Earliest" to "Latest"
6. Include suspicious ? Yes      Export ? Yes
7. Print or purge ?     Print and purge

Field number to change ? 
```

Parameters 1 through 6 have already been discussed thoroughly in our discussion of the Customer Cards listing program. Refer to that report for more information on them. We'll just cover the one additional parameter for this version of the program.

"7. Print or purge ?" This parameter allows you to print the listing without deleting any records (basically creating a listing), purge the selected records without creating a listing at all or creating a report of the records that were purged. Generally I recommend the latter so you have an audit trail. But this report might carry sensitive credit card information so you should take that into consideration. Run the report in "Print only" mode first to ascertain what records would be deleted and what information appears on the report. Make your decisions from there. If you do create a report containing sensitive information you should probably "Print to disk", physically print the report then delete the report from the spooled reports.

The second parameter screen filters the records on their position in the customer cards definition table as it appears in the *Setup>Customers>Control>"Customer cards"*. This allows you to purge the cards that have been assigned only by their "slot" in the table. It's less useful to you as a listing, but it's very useful for purging because it allows you to purge all of your CP-Gateway customer cards in a single pass based on the slot(s) they take in the customer cards table.

Here's the "by position" parameter screen.

```

Purge customer cards 7.5.20

Please enter:

1. Customer-# range      "First"      to "Last"

2. Customer card types
   Discover      Y      7      Discover      Y
   EMU1         Y      15     EMU Paymnt    N
   EMU2         Y      15     EMU Paymnt    N
   EMU3         Y      15     EMU Paymnt    N
   EMU4         Y      15     EMU Paymnt    N
   EMU5         Y      15     EMU Paymnt    N
   EMU6         Y      15     EMU Paymnt    N
   CustCard     Y

3. Include suspicious ? "All"      Export ? Yes

4. Print or purge ?      Print and purge

Field number to change ? 

```

Parameters 1 through 3 have already been discussed thoroughly in our discussion of the Customer Cards listing program. Refer to that report for more information on them. We'll just cover the one additional parameter for this version of the program.

"4. Print or purge ?" This parameter allows you to print the listing without deleting any records (basically creating a listing), purge the selected records without creating a listing at all or creating a report of the records that were purged. Generally I recommend the later so you have an audit trail. But this report might carry sensitive credit card information so you should take that into consideration. Run the report in "Print only" mode first to ascertain what records would be deleted and what information appears on the report. Make your decisions from there. If you do create a report containing sensitive information you should probably "Print to disk", physically print the report then delete the report from the spooled reports.

The ability to purge suspicious records and export them is particularly useful when cleaning trashy data from your customer cards. If you indicate to include "Only" suspicious records and indicate to export these records you purge the trashy records from the file and provide a way to get the records back into the file. Purging the records is very important if you intent to correct them and put them back in the file.



Let's say you indicate to include "Only" suspicious records and indicate to export these records. But at the "7. Print or purge" parameter you select "Print only". The trashy records stay in the file. You then spend a considerable amount of time to correct the records including records with duplicate customer card definition index values. Now you use the Customer file utilities to import the corrected records. Hopefully you select "Replace records in file" as you do or you'll fix nothing. But those records whose indexes you corrected get added to the file – they don't replace the trash record you corrected because the index field is one of the record's primary key fields. So the primary key for the corrected record is different from the original trashy record's key and so is a different record to the file system. Now you not only still have the original record which the trash record had a duplicate index with; you also have the original trash record you exported AND you now have the corrected record you just imported. You just made the situation worse.

The report's format is the same regardless of the parameter screen used. If suspicious records are being exported the record number of the suspicious record is included on the second detail line. See the end of this document for an [example](#) of the resulting report.

## Customers>Utilities>Open item maintenance

This existing program allows maintaining the open item file. It allows changing fields that are part of the record's primary key which means the record cannot merely be rewritten to disk. The original record must be deleted and the new record written. Because the open item's primary key is also an index to the EMV transaction history file we need to rewrite the EMV record with the new open item's primary key if that key changes. We changed this program to do just that.

We added a couple of new EMV-related fields to the open item (and closed item) files. These fields are normally maintained by our modifications to CounterPoint programs. But since this program allows creating and modifying record directly, we needed to added these new field to this program. Below is a screenshot of the revised program.

```
Open items 7.5.20

1. Customer #      A10           John Ross
2. Document date  11/12/20
3. Document #     000103
4. Document type  Payment
5. Apply-to #    * Open *

6. Terms code     <Not applicable>
7. Due date      11/12/20
8. Amount        51.00
9. Other amount   .00           Balance: 51.00

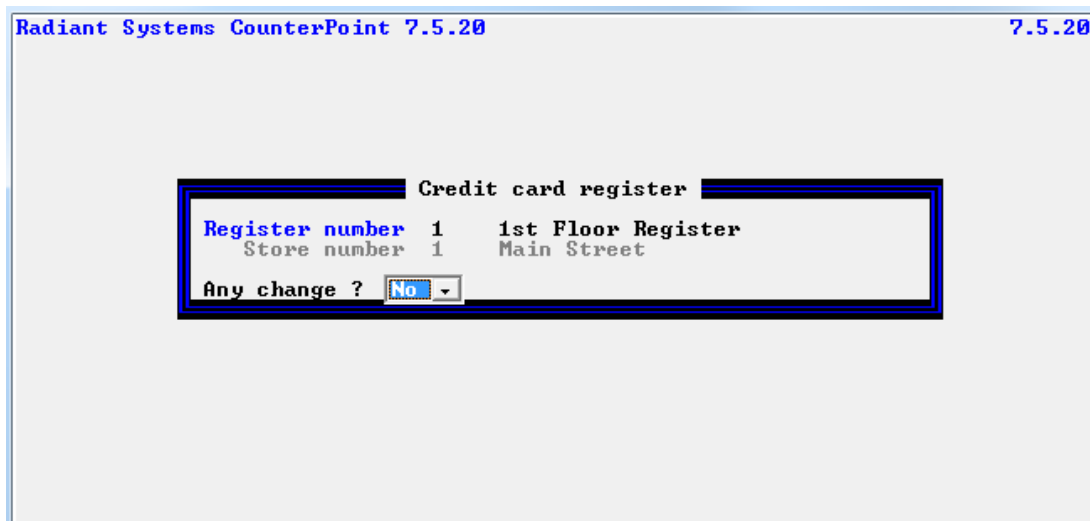
10. P.O.#
11. Order #
12. Reference
13. Sales rep    023           Margaret Rockwell
14. Store #
15. EMU Payment ? Y
16. EMU RequestId 01Z6ILR9TI01U680G4EH2LIHC18TJS3W

Field number to change ? 
```

## Order Entry>Orders>Enter

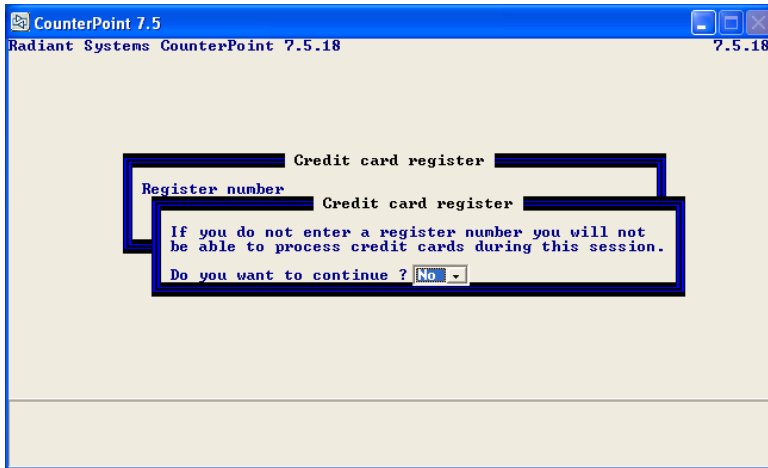
We added the ability to perform EMV payments to Order Entry. The process works pretty much the same as before, but there are a few enhancements. We also added the ability to use a customer's card for both deposits and final payments. Vanilla CounterPoint never had that.

Vanilla CounterPoint communicates directly with the credit card terminal. It controls the terminal and sees all the raw, unencrypted information on the credit card. That's one of the liabilities that CP-EMV eliminates. CP-EMV employs FreedomPay's Figaro service to communicate with the Ingenico. Since Figaro uses a workstation-id to map the currently running instance of CounterPoint to a particular Ingenico, if you intend to utilize an Ingenico to accept credit cards in Order Entry you need to specify the workstation-id you'll be using to swipe/dip the credit cards. The value you've assign to the field *Setup>Order Entry>Control>"21. Default credit card reg"* controls whether you are prompted for a register number. If you provided a register number in this field the program will use that register number and will not prompt for a register number. Likewise, if you set the field to "Not used" the program understands Order Entry doesn't use an Ingenico and so won't prompt for a register number. Otherwise the program prompts the operator for the register to be used during the session. Here is a sample screen of that.



The operator enters the register number associated with the Ingenico he intends to use during the session. The program then displays the store info the register is associated with and a literal indicating whether the store is using CP-Gateway or FreedomPay for credit card processing. The register is then locked from being used by another workstation in the same way Point of Sale does. If another workstation attempts to open the same register, that operator will receive the ubiquitous "Register in use at another station" message. That prevents two workstations trying to send authorization requests to the same Ingenico. The same check is made if you have assigned a register number in *Setup>Order Entry>Control>"21. Default credit card reg"*. If that register is locked you'll receive the message "Default register in use at another station" and the program will prompt for a different register number.

If the operator doesn't enter a register number the program provides the following warning:



If the operator answers "N" the cursor returns to the register number field. If the operator answers "Y" the program continues to the entry screen in the normal manner. The wording of this second screen is a bit misleading, but we couldn't explain everything on a couple of lines. Basically, if you answer "N" you can still enter orders and even apply credit card pay codes to them. You can even get authorizations for tokenized customer cards. We merely left the message with the same verbiage as in the other areas of CounterPoint that prompts for the register number.

But there is a potential caveat in not providing a register number when prompted. By entering a register number the operator is also indicating the merchant-id to be used for the authorization. The merchant-id for the store to which the register is assigned is used. If the operator skips entering the register number the program reverts to using the store number that was assigned in the field "Setup>Order Entry>Control>"1. Store number". The same is true if Setup>Order Entry>Control>"21. Default credit card reg" has been set to "Not used". This is OK if you only have a single store or you're OK with all of your deposits being attributed to a single merchant-id. But if you have multiple merchant-ids and you need the authorizations to go to the correct ones, you need to set CP-EMV to prompt for the register number.

Vanilla CounterPoint allowed entering the credit card information, and authorizing the transaction immediately or in a batch later or during posting. By their nature, EMV cards must be processed "up front" because the system doesn't store the credit card information. So the authorization must take place in the transaction entry program.

O/E orders have two tender fields, one for deposits and one for the final payment. CP-EMV allows using tokens in lieu of a customer's credit card for both. The order maintenance program allows keying both. The deposits and refunds program only allows entering deposits. To keep everything in sync, the order maintenance program won't allow entering final payment information if there is an unposted deposit. You need to post the deposit before you can enter the final payment. Similarly, the deposits and refunds program will not allow entering a deposit on an order that has a final payment.

Here's a screenshot of CP-EMV's orders payments screen.

```

Orders (Enter) 7.5.20
*** Card on file ***
Order #: 100077  Cust #: 010  Loc: 1  Order date: 9/23/20
Type: Order      Terry Watson  Pick date: None
Terms: 2IN 2% 10 Net 30
Subtotal      30.60
1. Freight    .00
2. (Reserved)
3. Disc %    .000
4. Tax       2.68
Invoice total 33.28
Deposit applied .00
Amount due   33.28

5. # ship labels 0
6. # COD labels  0

7. Commission 2.45
8. Deposit
9. Final pmt

Pay-code Amount
None
Cash
Checks
MasterCard
Coupon
Uisa
Discover
Am Express
EMU1 Str#2
EMU2
Customer card
Weight

Ordered:  Lines  Quantity  Ext-price  Cost  Weight
To-ship:  1      1.000     30.60     10.54   .000
          1      1.000     30.60     10.54   .000

Field number to change ? 8

F7=Customer card
  
```

Notice at the top it has the literal "\*\*\* Card on file \*\*\*". When this screen first loads the program loads a table with the customer's tokenized cards. If the customer has any this literal displays. Also note that under the customer's name we display the terms. We added that as well. Basically we copied the code that displays the same field on the first screen.

While entering a deposit's pay code field the program displays the expected pay codes including the EMV pay code you've already defined. If the customer has a tokenized customer card on file the program also offers "Customer card" at the bottom of the list and <F7> to select one of those.

Pressing <F7> displays the customer's cards to select from like this.

```

Orders (Enter) 7.5.20
*** Card on file ***
Order #: 100077  Cust #: 010  Loc: 1  Order date: 9/23/20
Type: Order      Terry Watson  Pick date: None
Terms: 2IN 2% 10 Net 30
Customer cards
1. Card-name Card-# Exp. ..Pay-code... TknExpDt
2. EMU1 542418*****1765 1223 15 EMU2 9/17/22
3. EMU2 601100*****0009 1224 15 EMU2 9/17/22
4. EMU3 476173*****0010 1225 15 EMU2 9/18/22
   EMU4 371449*****8431 1225 15 EMU2 9/17/22
Amount due 33.28
9. Final pmt None

5. # ship labels 0
6. # COD labels  0

Ordered:  Lines  Quantity  Ext-price  Cost  Weight
To-ship:  1      1.000     30.60     10.54   .000
          1      1.000     30.60     10.54   .000

Field number to change ? 8
  
```

Selecting on auto-populates the pay-code, card# and expiration date fields and takes the cursor to the payment amount field.

```

Orders <Enter>                                     7.5.20
*** Card on file ***
Order #: 100077   Cust #: 010   Loc: 1   Order date: 9/23/20
Type: Order     Terry Watson   Pick date: None
Terms: 2TN 2% 10 Net 30
Subtotal        30.60   7. Commission  2.45
1. Freight      .00
2. <Reserved>
3. Disc % .000    .00
4. Tax          2.68
Invoice total   33.28
Deposit applied .00
Amount due     33.28

8. Deposit      Pay-code Amount
Card#          15 EMU
ExDt/Auth#    542418*****1765
              1223

9. Final pmt   None

5. # ship labels 0
6. # COD labels  0

Ordered:  Lines  Quantity  Ext-price  Cost  Weight
To-ship:  1      1.000    30.60     10.54  .000

```

Enter an amount. (We're cheap. We'll put in a dollar.) The amount is displayed and the cursor goes to FNTC and you're back to vanilla CounterPoint code. But at this point the order has everything it needs to authorize the deposit. You don't need to authorize it though. The token has been saved on the order so you can save the order and authorize it later. When you press <ENTER> at FNTC the program asks "Authorize deposit now?". The default is "Yes", but you can change it to "No" and continue. Or you can accept the "Yes", the deposit will authorize and the screen will close. If the authorization fails you stay at FNTC. You can fix the problem and try the authorization again or you can press <ENTER> and FNTC (again) this time answering "No" to the "Authorize deposit now?" prompt and deal with it later.

Entering the final payment work exactly the same way so there's no point in repeating all that verbiage. The only change is when you assign a tokenized customer card then press <ENTER> at FNTC the program asks "Authorize balance now?".

To better depict the next topic I added a couple more deposits to the same order totaling six dollars (still cheap). Pulling the order back up on the screen (the payments screen) you see the program now offers <F3> for "EMV Records". Pressing <F3> calls the portable EMV history transaction viewing program. That displays the EMV history for this order as shown below.

```

Orders <Enter>                                     7.5.20
Order #: 100077      Cust #: 010      *** Card on file ***      Loc: 1      Order date: 9/23/20
Type: Order          Terry Watson      Pick date: None
Terms: 2TN 2% 10 Net 30

                    Payments
1. >
2. <Deposit> 1.00 Card#/Exp 542418*****1765 / 12-12
3. >          Sale   Date/Auth 11/10/20 / 13:10:26 / 684957
4. >          Request-Id 01Z61LM3NF01U68007DCMJHRQIRFFS13
   <Deposit> 2.00 Card#/Exp 601100*****0009 / 12-12
   Sale   Date/Auth 11/10/20 / 13:31:36 / 685184
   Request-Id 01Z61LM4U001U68008E637QGDBV0U075
   <Deposit> 3.00 Card#/Exp 476173*****0010 / 12-12
   Sale   Date/Auth 11/10/20 / 13:37:11 / 685232
   Request-Id 01Z61LM59F01U68008027P95MRCHOS7W
5.
6. # COD labels      0

Ordered:  Lines      Quantity      Ext-price      Cost      Weight
To-ship:  1          1.000          30.60          10.54          .000

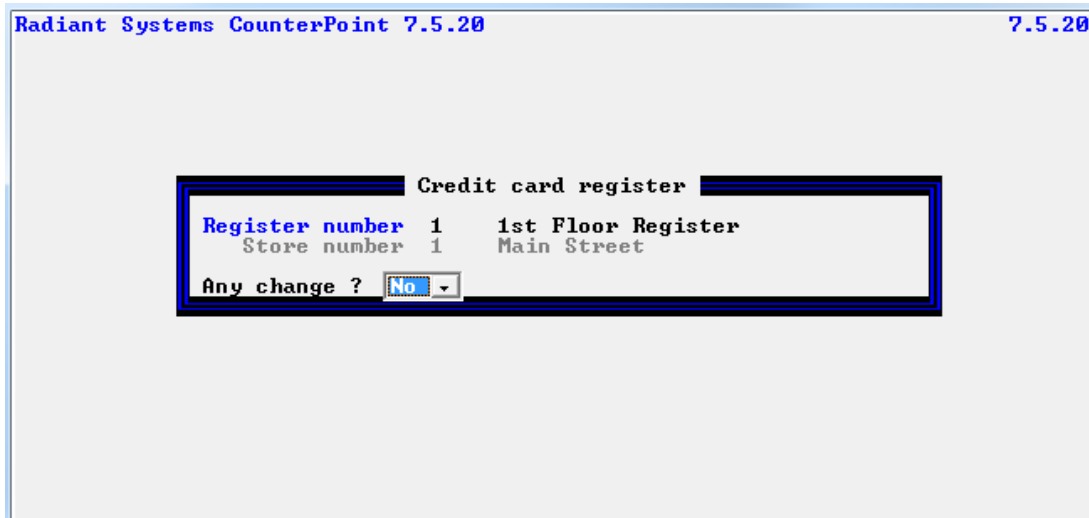
```

We added three deposits for the order and they are displayed. The list displays the EMV tenders in “greenbar” format (remember that paper ?) alternating between grey and black. The currently selected payment’s background is blue. You can arrow up and down the list and press <ESC> to exit. If there was a final payment for this order it would display as well.

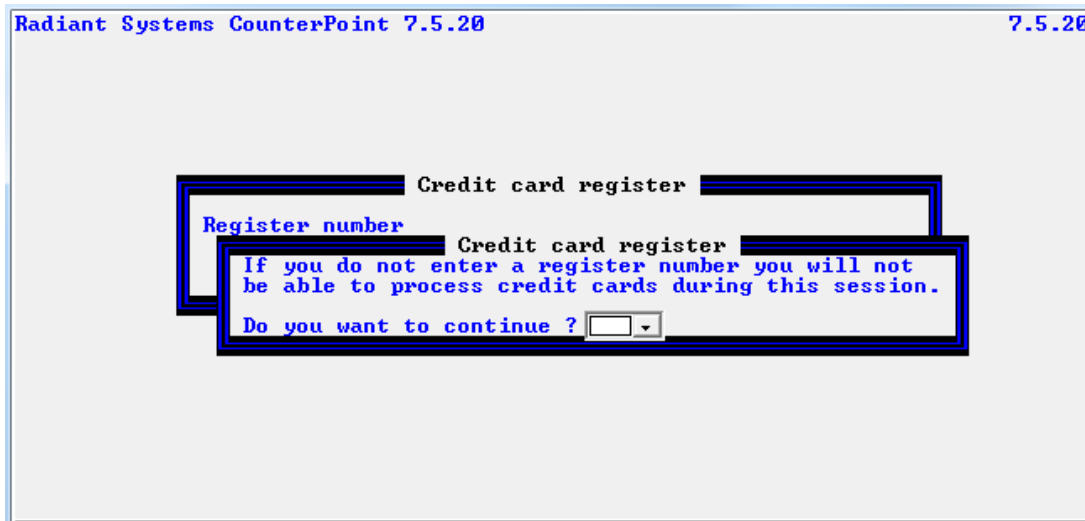
## Order Entry>Deposits and refunds>Enter

We added the ability to take EMV payments as well as the ability to employ customers' tokenized customer cards. Tendering a credit card works pretty much the same as before except for a couple of changes.

Vanilla CounterPoint communicates directly with the credit card terminal. It controls the terminal and sees all the raw, unencrypted information on the credit card. That's one of the liabilities that CP-EMV eliminates. CP-EMV employs FreedomPay's Figaro service to communicate with the Ingenico. Since Figaro uses a workstation-id to map the currently running instance of CounterPoint to a particular Ingenico, if you intend to utilize an Ingenico to accept credit cards in Deposits and refunds you need to specify the workstation-id you'll be using to swipe/dip the credit cards. The value you've assign to the field *Setup>Order Entry>Control>"21. Default credit card reg"* controls whether you are prompted for a register number. If you provided a register number in this field the program will use that register number and will not prompt for a register number. Likewise, if you set the field to "Not used" the program understands Order Entry doesn't use an Ingenico and so won't prompt for a register number. Otherwise the program prompts the operator for the register to be used during the session. Here is a sample screen of that.



If the operator doesn't enter a register number the program provides the following warning:



If the operator answers "N" the cursor returns to the register number field. If the operator answers "Y" the program continues to the entry screen in the normal manner. The operator can still enter deposits and even apply credit card pay codes to them. They just won't be able to authorize those transactions. However, if the operator uses a customer card to tender the deposit he will be able to authorize the payment since using a customer card bypasses the need for an Ingenico.

But there is a potential caveat in not providing a register number when prompted. By entering a register number the operator is also indicating the merchant-id to be used for the authorization. The merchant-id for the store to which the register is assigned is used. If the operator skips entering the register number the program reverts to using the store number that was assigned in the field "Setup>Order Entry>Control>"1. Store number". The same is true if Setup>Order Entry>Control>"21. Default credit card reg" has been set to "Not used". This is OK if you only have a single store or you're OK with all of your deposits being attributed to a single merchant-id. But if you have multiple merchant-ids and you need the authorizations to go to the correct ones, you need to set CP-EMV to prompt for the register number.

The program "drives" pretty much as before with a few exceptions for handling credit cards. Enter the transaction type as usual. At the pay code field you'll be offered the usual pay codes including the EMV pay code you've already defined for the O/E store. If the customer has tokenized customer cards on file you'll also find "Customer cards" in the list of available pay codes plus the option to press <F7> for "Customer cards". (If the customer only has a single customer card the prompts are singular – "Customer card".)



Here's a sample screen.

```

Deposits and refunds 7.5.20

1. Order #      100074      Order type:    Order
   Order date   9/15/20      Status:       Open
   Customer #   400           Subtotal:     23.00
   Wilson & Wilson      Freight:      .00
   231 Lake Drive      Misc:         .00
   Memphis          TN 38901     Discount:     .00
                                       Tax:          2.01
                                       Invc total:   25.01

2. Trx type     Deposit
3. Pay code     Cash
4. Deposit amt  0.00         Dep applied:  12.00
                                       Amount due:   13.01

Field number to c
EMU2
Customer card

F7=Customer card
  
```

Select the desired pay code (EMV for us please) or press <F7> to select a customer card. If you press <F7> you'll be presented a window of the customer's tokenized cards to select from like this.

```

Deposits and refunds 7.5.20

1. Order #      100077      Order type:    Order
   Order date   9/15/20      Status:       Open
   Customer #   400           Subtotal:     33.28
   Wilson & Wilson      Freight:      .00
   231 Lake Drive      Misc:         .00
   Memphis          TN 38901     Discount:     .00
                                       Tax:          2.01
                                       Invc total:   33.28

2. Trx type     Deposit
3. Pay code     Cash
4. Deposit amt  .00         Dep applied:  17.00
                                       Amount due:   16.28

Customer cards
Card-name  Card-#  Exp.  Pay-code  TknExpDt
EMU1      542418****1265  1223 15 EMU2      9/17/22
EMU2      601100****0009  1224 15 EMU2      9/17/22
EMU3      476173****0010  1225 15 EMU2      9/18/22
EMU4      371449****8431  1225 15 EMU2      9/17/22
  
```

Select the desired card and the pay code, card number and expiration date field populate and the cursor goes to the payment amount. Enter that.

Unlike the order maintenance program, this program doesn't ask if you want to authorize the tender when you leave FNTC. Instead it offers <F8> to authorize the tender. But you don't have to immediately. You can save the deposit unauthorized and do it later.

Before the transaction is authorized you can change the pay-type and amount fields. There are also several possible function keys available.

Deposits and refunds		7.5.20	
1. Order #	100075	Order type:	Order
Order date	9/23/20	Status:	Open
Customer #	N10	Subtotal:	23.00
	Nancy Smith	Freight:	.00
	456 East Main	Misc:	
	Memphis TN 38111	Discount:	.00
		Tax:	2.01
		Inv total:	25.01
2. Trx type	Deposit	Dep applied:	3.00
3. Pay code	15 EMU2	Amount due:	22.01
4. Deposit amt	2.00		

Field number to change ?

F1=Next	Ctrl-F1=Previous	F2=Clear	F3=EMV records	F5=Cust cards
F8=Auth EMV payment				

The <F1> and <CTL+F1> keys are the usual navigation keys. Pressing <F2> will clear fields 2 through 4 – basically erasing the deposit. “F3=EMV records” is offered if the order has any EMV history on file. Pressing <F3> calls the portable EMV history viewing program to view the EMV history records associated with the order. “F5=Cust cards” is offered if customer cards is enabled and set up to carry EMV cards. Pressing <F5> opens the customer card maintenance program to allow editing the customer’s customer cards. One could use this key to add one (or more) customer cards for the customer while entering the deposit. That would be especially handy if you haven’t set CP-EMV up to auto-update customer cards. If the deposit is a credit/debit card payment and hasn’t been authorized yet, the program offers <F8> to authorize the tender.

When you do authorize the tender the screen will display a countdown while waiting for the authorization. Once the authorization is received the authorization code and the request-id is displayed. This gives an obvious indication the deposit has been authorized.

Deposits and refunds		7.5.20	
1. Order #	100074	Order type:	Order
Order date	9/15/20	Status:	Open
Customer #	400	Subtotal:	23.00
	Wilson & Wilson	Freight:	.00
	231 Lake Drive	Misc:	
	Memphis TN 38901	Discount:	.00
		Tax:	2.01
		Inv total:	25.01
2. Trx type	Deposit	Dep applied:	15.00
3. Pay code	15 EMU2	Amount due:	10.01
4. Deposit amt	3.00		
Card#	476173*****0010 Exp:12/23		
AuthCode	760920		
Req-id	01Z61LP6HK01U6808EAFEEU61C4FBKJ1		
Field number to change ? <input type="text"/>			
<div style="display: flex; justify-content: space-between; border-top: 1px solid black; padding-top: 5px;"> <span>F1=Next</span> <span>Ctrl-F1=Previous</span> <span>F2=Clear</span> <span>F3=EMV records</span> <span>F5=Cust cards</span> </div>			

Once it's been authorized you cannot merely change any of the fields. Attempting to will result in the message "Transaction already authorized" and the cursor remains at FNTC. If you really want to change the transaction you'll need to void the prior authorization. Upon pressing <F2> to clear the existing payment the program will ask "Clear previous authorization?". Answering "N" takes you back to FNTC. Answering "Y" will cause the program to attempt voiding the transaction. If it is successful you'll get the message "Successfully voided". If too much time has passed since the tender was authorized the void will fail. But the program won't give up yet. The EMV record of the authorization is still on file and on that is the token that was returned with the authorization. The program can use that as a placeholder for the customer's card to generate a return. So then the program asks "Do you want to perform a return using the token?". Answer "N" and the cursor returns to FNTC. Answer "Y" and the program attempts to make the return. If that works you'll get the message "Return authorized". If it doesn't you only have a single option left. If you have an Ingenico available and you indicated a register number when you started this program (or you a default register assigned in the O/E control record) you be asked "Do you want to perform a return via the Ingenico?". Answer "N" and the cursor returns to FNTC. Answer "Y" and the program will wake up the attached Ingenico and prompt for a card. You can swipe the card or enter the card number on the Ingenico (if it has been set up to allow it).

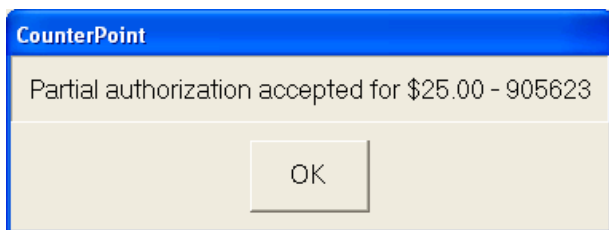
Deposits and refunds		7.5.20	
1. Order #	100077	Order type:	Order
Order date	9/23/20	Status:	Open
Customer #	010	Subtotal:	30.60
	Terry Watson	Freight:	.00
	5832 Misty Oak Drive	Misc:	
	Memphis TN 38138	Discount:	.00
		Tax:	2.68
		Inv total:	33.28
2. Trx type	Deposit	Dep applied:	17.00
3. Pay code	15 EMU2	Amount due:	16.28
4. Deposit amt	11.00		
Card#	411112*****2010	Exp:	06/16
AuthCode	686787		
Req-id	01261LMAMA01U6800DR1CIS0NQAOR4HO		

Field number to change ?

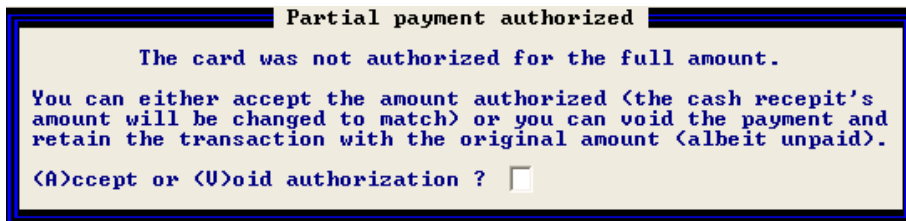
F1=Next   Ctrl-F1=Previous   F2=Clear   F3=EMV records   F5=Cust cards

An authorized transaction displays the authorization code, the masked card number and the EMV transaction's request-id.

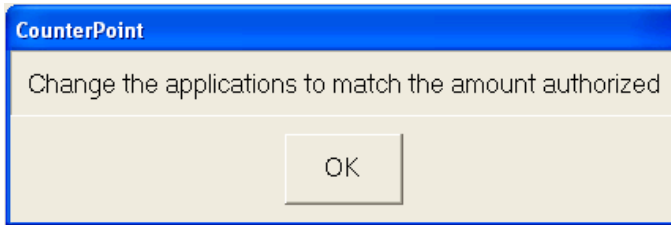
If the authorization returns a partial payment an appropriate message displays notifying the operator of the partial payment (playing the partial payment sound if it's set up) and displaying the authorized amount and the auth code.



Then the following message is displayed.

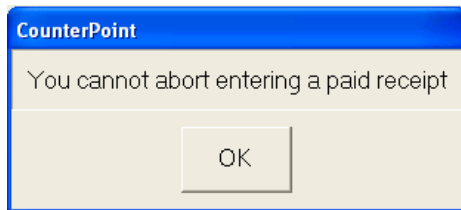


If the operator answers "Void" the program will void the authorization and the cursor will return to FNTC. If the operator answers "Accept" the cash receipt's amount field is automatically changed to the amount authorized. If the customer is an open item customer the amount(s) applied no longer match the amount received so the program displays another message.



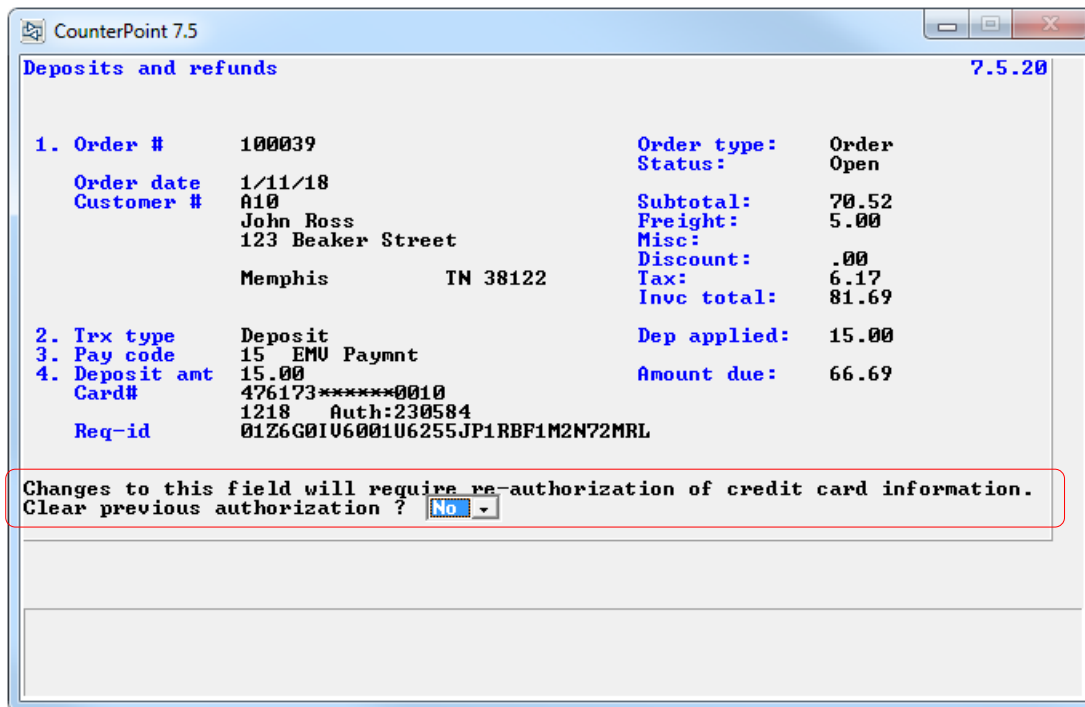
At this point the operator needs to modify the apply-to fields as required to match the amount that was actually authorized. The message will redisplay every time the operator presses <ENTER> at FNTP until the apply-tos match the amount authorized.

Note that attempting to abort entering a paid (via EMV payment) will result in the message:



Thus an authorized transaction must be completely entered. If the operator wants to cancel the transaction after it has been authorized he will need to complete the transaction, pull it back up in change mode and delete it there. When an EMV-tendered deposit is deleted the payment is automatically voided.

If the operator attempts to change any field that touches the deposit's payment the program displays a warning message like below.



Answering “Yes” in the unmodified program merely clears the payment fields from the screen and allows the operator to continue. But CP-EMV has already authorized the existing payment. So answering “Yes” here forces the CP-EMV version of the program to immediately attempt to issue a return for the payment utilizing the token that was returned with the original authorization. That should always work and the operator will receive an “Authorization successful” message (if successful authorization messages are turned on). No customer card will be prompted for. However, if the return transaction failed for some reason the program will try to generate the return by prompting for a credit card. Dipping/swiping/waving the customer’s credit card (or manually entering the card information) will get the return transaction created. If even that fails, the program won’t allow changing the field and the cursor will return to the field that was being changed having not accepted the changed value.

Unauthorized transactions are skipped over in the posting program until they are authorized. In the Edit List, unauthorized transactions display the literal “\*\*\* Credit card needs to be processed \*\*\*”. Authorized transactions update the new EMV information file.

### **Order Entry>Deposits and refunds>Edit list**

The vanilla report printed a string of asterisks to represent the card’s expiration date. Our version does the same if there’s an expiration date on the record. If there isn’t the deposit might not have been authorized. If that’s the case the report now displays “\*\*\*\* Not authorized \*\*\*\*” instead of the expiration date. If the payment was authorized via EMV and the payment was in home monetary units, the program now displays the request-id of the authorization in the Validate-2 field. So nothing big.

### **Order Entry>Invoices>Authorize**

Once you’ve switched to CP-EMV you’ll never go back. I mean you’ll never need to deal with CounterPoint’s EDC stuff anymore. CP-EMV makes this entire program obsolete so we kicked it to the curb (programmatically speaking of course).

## System>View>EMV Transaction history

Similar to other areas of CounterPoint we've created a program that allows viewing EMV transaction history records. This allows you to inquire specific EMV records without having to run a report. Even better, you can view the signature associated with the transaction (if one exists) and if you're a manager you can even void a transaction. Of course that's a lot of power so be careful with that last superpower.

The upper left corner of the screen has four parameter fields to filter the results displayed. Their purpose is actually pretty obvious. If you're not authorized to view records outside your current store number the program won't allow you to enter anything else. The starting date parameter seeds the date for the first record displayed. But once records are displayed you can actually view records before that date entered. This parameter merely starts you off at a particular date in the file. The record-type and transaction-type filters restrict the records displayed to whatever you select. Below is a sample screen.

View EMU transactions							7.5.20
Store-#	"All"	Customer: A10		John Ross			
Starting date	"Earliest"	Date:	11/12/20	Doc#:	000103	Seq#:	
Trx type	"All"	Typ:	P	Apply-to:		Seq#:	
Record type	"All"						
Store	Date	Time	RecTyp	TrxTyp	Amount	Card-number	
1	10/28/20	19:38:24	Order	Return	25.01-	401200*****0026	
1	10/28/20	19:38:55	Order	Void	25.01-		
1	10/28/20	19:39:04	Order	Return	25.01-	476173*****0010	
1	11/10/20	13:10:26	Order	Sale	1.00	542418*****1765	
1	11/10/20	13:31:36	Order	Sale	2.00	601100*****0009	
1	11/10/20	13:37:11	Order	Sale	3.00	476173*****0010	
1	11/10/20	13:51:24	Order	Sale	1.00	476173*****0010	
1	11/10/20	13:52:58	Order	Sale	2.00	476173*****0010	
1	11/10/20	15:09:05	Order	Sale	11.00	411112*****2010	
1	11/11/20	17:16:06	Order	Sale	3.00	476173*****0010	
1	11/11/20	18:13:25	Order	Void	.00	815017*****2690	
1	11/11/20	18:31:23	Order	Sale	1.00	476173*****0010	
1	11/11/20	18:40:43	PosTkt	Sale	10.00	411112*****2010	
<b>1</b>	<b>11/12/20</b>	<b>12:26:38</b>	<b>OpnItm</b>	<b>Sale</b>	<b>51.00</b>	<b>476173*****0010</b>	<b>\$</b>

F1=EmvInfo   F2=TagInfo   F3=SigCap   F5=Payments   F8=Void   ESC=Exit

The upper right corner of the screen has a box that displays the details that tie the entry to a specific CounterPoint transaction. For some record types there isn't much there, but generally there's enough to associate the entry quite well.

Once records are on the screen you'll have one or more function keys available to zoom additional information, view the signature or void the transaction. All entries offer <F1> to view additional EMV transaction information like this.

```

View EMU transactions 7.5.20
Store-# "All"
Starting date "Earliest"
Trx type "All"
Record type "All"
Customer: A10 John Ross
Date: 11/12/20 Doc#: 000103 Seq#:
Typ: P Apply-to: Seq#:

EMU transaction information
CardType: credit Issuer: VISA
NameOnCard: Test/Card 01 ApproveAmt: 51.00
MaskCardNo: 476173*****0010 ExpDt: 12/18 RequestAmt: 51.00
AuthCod: 865654 Dec: A ErrCod: 100 PartialAmt: .00
EntryMode: icc Sig: Y Offline: N AcctBal: .00
LaneId: DCC: N CUM: FloorLimit: 50.00
MerchantRefCod: Z-A10-10 TipAmt: .00
TransactionId: 0000000000000010941
RequestId: 01Z6ILR9TI01U680G4EH2LIHC18TJS3W

1 11/11/20 17:16:06 Order Sale 3.00 476173*****0010
1 11/11/20 18:13:25 Order Void .00 815017*****2690
1 11/11/20 18:31:23 Order Sale 1.00 476173*****0010
1 11/11/20 18:40:43 PosTkt Sale 10.00 411112*****2010
1 11/12/20 12:26:38 OpnItm Sale 51.00 476173*****0010 S
  
```

Sales and returns also have tag data that is returned with the authorization. Pressing <F2> displays those fields like this.

```

View EMU transactions 7.5.20
Store-# "All"
Starting date "Earliest"
Trx type "All"
Customer: A10 John Ross
Date: 11/12/20 Doc#: 000103 Seq#:
Typ: P Apply-to: Seq#:

EMU tag information
50: UISA CREDIT 9F0E: 0010000000 DF04:
5F2A: 840 9F0F: F040009800 DF05:
5F34: 01 9F10: 06010A0360BD00 AID: A0000000031010
82: 5C00 9F12: UISA CREDIT ARC: 00
95: 42C0008000 9F1A: 840 IAD: 06010A0360BD00
9A: 201112 9F26: 65DE28C06B27FFA4 TSI: E800
9C: 00 9F27: 40 TUR: 42C0008000
9F02: 9F34: 1E0300 TAC-DEFAULT:
9F03: 0.00 9F36: 03C2 TAC-DENIAL:
9F07: FF00 9F37: 676A975A TAC-ONLINE:
9F0D: F040008800 DF03:

1 11/11/20 18:13:25 Order Void .00 815017*****2690
1 11/11/20 18:31:23 Order Sale 1.00 476173*****0010
1 11/11/20 18:40:43 PosTkt Sale 10.00 411112*****2010
1 11/12/20 12:26:38 OpnItm Sale 51.00 476173*****0010 S
  
```

If a signature file is found for the entry an upper-case "S" character is displayed on the far right edge of the line and <F3> is offered to view it. For more info on the signature capture feature see [HERE](#).

The entries are sorted by date and time. While looking at an entry you may want to see the entry in relation to the other EMV records that are related to the transaction. Pressing <F5> invokes the portable EMV history viewing program that displays all EMV records for a transaction. So while you're viewing an EMV transaction you can see all the EMV activity for the transaction with a keystroke. See [HERE](#) for more information on that feature.



If the entry is a sale or a return, the program offers <F8> to void the authorization. This is a powerful feature and its effect are permanent so be careful how you use it. Transactions that are voided here are immediately and permanently voided. No recourse. Also the matching transaction in CounterPoint is not deleted. So if you have a ticket with credit card tender on it and you void the authorization here, you still have a ticket in POS that will post and relieve inventory, reflect the sale in history and yet you won't get paid for it. When you press <F8> the program will ask "Do you want to void this transaction?". If you answer "Y" the program attempts the void. If it fails the program asks "Transaction could not be voided. Return instead?". If you answer "Y" the program uses the token that was returned with the original authorization to perform a return against the sale. (So no Ingenico is required.) If that fails the program displays "Transaction not voided or returned".

This program can display the EMV history for all stores. If one were to void a transaction it doesn't matter what store they're in. The transaction is backed out like it never happened. But if a return is performed instead the returned funds should go back to the originating store. That means the program uses the store number from the transaction for the merchant credentials to send with the return transaction. But what if the originating store record is no on file? When you first attempt to void a transaction, the program attempts to read the originating store's record. Normally that works just fine. But if it's not there the program presents the message "Trx's store rec missing - Substituting store 999" and the void process continues. You are not offered a chance to stop the void. That's because the voiding process eliminates the original transaction. It doesn't send money anywhere so whether the originating store record is on file or not doesn't matter. But the message serves as a warning in the event the void fails. If that happens the program will offer to make a return instead. That return will move money to the store whose credentials are used. If the originating store is on file, the monies will be applied to the originating store. If the originating store is not on file the monies will go to the store of the person voiding the EMV transaction. That's going to throw things off. So if you get that message warning you the originating store is not on file you should pause to consider the ramifications of answering "Yes" to performing a return instead.

If you void or perform a return against a displayed transaction the screen refreshes to include the new transaction.

If you've been using CP-EMV for a while you may notice that void type transactions have zeros in the amount column. That's because we didn't notice that when FreedomPay returns the results from a void they clear the amount field being returned. We corrected this in the version 4.0x upgrade but this doesn't automatically correct the data.

But you can fix this if you're running Windows (and have Pervasive Control Center installed) or have TransSoft SQL installed with an application that can connect to your EMV transaction table and run an SQL statement. The idea is to set the approved-amount field to the requested-amount field on void type transactions where the approved-amount is currently zeros. Here's the PCC version of that SQL statement:

```
update SY_EMV set APPROVED_AMT=REQ_AMT where TRX_TYP='V' and APPROVED_AMT=0
```

If you're not using PCC to run the statement you might need to tweak it slightly to make it work.

## System>Reports>EMV Transaction history

There will probably come a time when you need to inspect/review/audit individual historical EMV There will probably come a time when you need to inspect/review/audit individual historical EMV transactions. Every successful EMV transaction (except creating a token) creates an EMV transaction record. We retain every scrap of information returned from FreedomPay (except for an image of the signature) in this file. The EMV Transaction history report provides a way to view this information.

Here is the parameter screen.

```
EMV Transaction history report 7.5.20

Please enter:

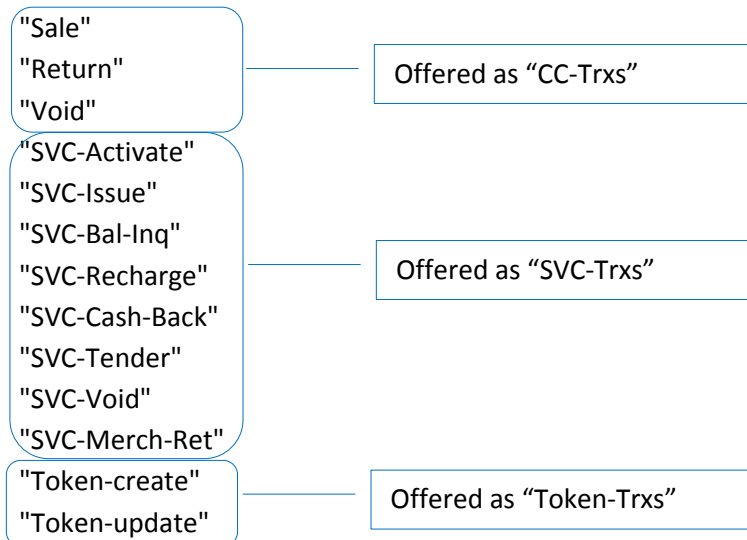
1. Store-#           "All"
2. Date range       "Earliest" to "Latest"
3. Transaction type "All"
4. Record type      "All"
5. Report format    Full <EMV tag data>
6. In order by      Store#/date

Field number to change ? 
```

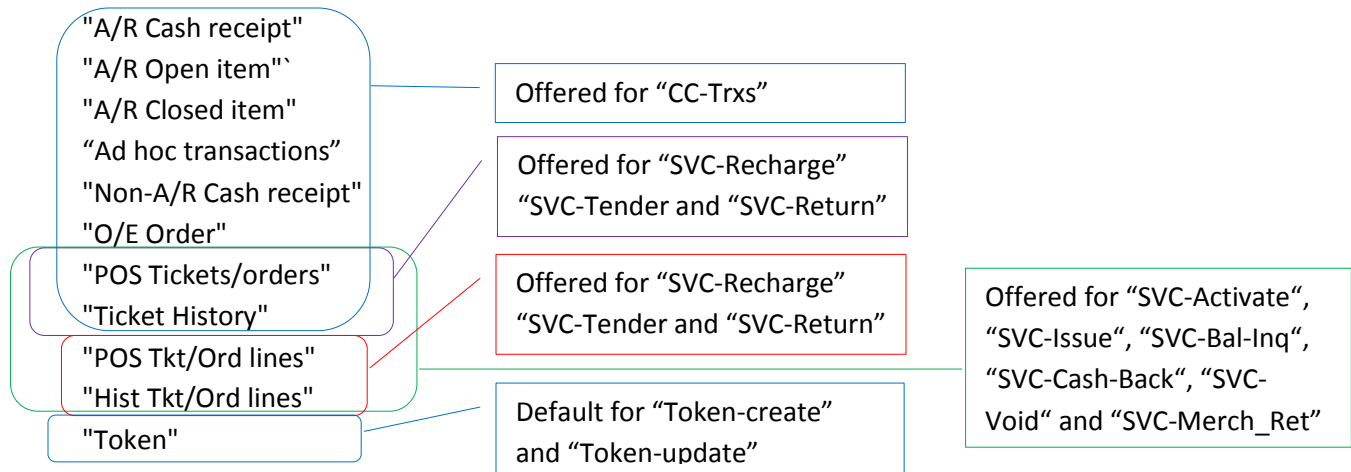
The “**Store-#**” parameter filters the records reported to those that originated in the store number provided. The parameter allows for <F1> for “All” and <F9> for DataLookup.

The “**Date range**” parameter is self-explanatory.

The “**Transaction type**” parameter is a dropdown box which offers each of the possible EMV transaction types. It also offers some of the transaction types grouped together so you can view all of the SVC transactions by themselves (for example). It also offers <F1> for “All”. The offered values are:



The **“Record type”** parameter refers to the area within CounterPoint where the record was generated. Not all of the possible values make sense for any specific transaction type. So all of the possible values for this field are only offered if you selected **“All”** for the transaction type parameter. Below is a list of the possible values grouped by the transaction types that allow them.



The **“Report format”** parameter allows 4 values – **“Micro”**, **“Summary – 1 line”**, **“Brief – 3 lines”** and **“Full (EMV Tag data)”**. The first line of all four formats displays identifying information for the transaction. This information varies depending on the record type of the record. The identifying fields associated with an A/R transaction will be different from a ticket history transaction. No EMV information is displayed on the first line. The other lines (detail lines other than the first line) are common to all record types. For this reason the program automatically groups all transactions by record type.

**“Micro”** format prints a single line for each EMV record. When using this format the report limits you to sort by **“Date”** or **“Store#/Date”** (the parameter described below). The first page of the report is dedicated to printing legends and headers. The field that uniquely identified transactions within CounterPoint is named **“Record-id”**. The field is a composite of the various fields that comprise the record’s primary key separated by period characters. (We chose periods because hyphens often occur in document numbers.) This first page provides the legend that describes the fields that comprise the record-id field.

**“Summary (1 line)”** prints a single line for each CounterPoint transaction information followed by a single line for each EMV transaction associated with the CounterPoint transaction.

**“Brief (3 lines)”** and **“Full (EMV Tag data)”** formats display one line for each CounterPoint transaction detail. The next 2 or 3 lines display EMV transaction information such as the request-id, amount, authorization code and such.

**“Full (EMV Tag data)”** displays the same fields as **“Brief (3 lines)”** plus another 8 lines of EMV tag data in addition to the 3 described above. If you want to kill a lot of trees this is the format to use. A tokenizing transaction does not return any EMV data. (In fact the process prompts to swipe the card – not dip.) So if you select a **“Token”** type at the **“Transaction type”** parameter **“Full”** will not be offered for this parameter.

See the end of this document for a [sample](#) of all three formats of this report.

## System>Reports>EMV Reconciliation report

Prior to using CP-EMV you probably had a separate pay code set up for each type of credit card tendered. That allowed several different ways to report the moneys associated with each at the end of the month when you reconciled your merchant statement. You can still set up different pay codes for each credit/debit card type using CP-EMV, but probably don't want to since your clerks probably won't be handling your customers' cards and so won't know which type of card is being tendered.

To help reconcile your merchant statement we created the EMV Reconciliation report. This report sorts the EMV history file by the "Issuer name" field within store# which should group the statement data in a way to help accumulating the activity by card type. Below is the parameter screen.

```
EMV Reconciliation report 7.5.20

Please enter:

1. Store-#      "All"
2. Date range   "Earliest" to "Latest"
3. Issuer       "All"
4. Report format Full

Field number to change ? 
```

The "**Store-#**" parameter prompts for the store number associated with the EMV transaction. In all areas of CounterPoint that allows EMV authorizations the program prompts the operator for a register number to use. (This allows CP-EMV to know which Ingenico to wake up.) The store number assigned that register is saved with the EMV transaction information record. If the operator is restricted to the current store/location, the program will default this parameter to that value. If the operator changes the value the program will display an error message and return to the store number parameter.

The "**Date range**" parameter is self-explanatory.

The "**Issuer**" parameter is a dropdown box that offers "AMEX", "DISCOVER", "MASTERCARD", "VISA" and "All". We arrived at these values during testing, once the product went "live" we quickly discovered that the issuer field returned with authorizations can have many, many more values than these. We suggest that you run the report in "All" mode to let the program report all possible values unless you know the specific value that you need.

The "**Report format**" parameter allows "Brief" and "Full". "Brief" only reports the total associated with each issuer reported. "Full" displays a single line for each transaction with a total line for each issuer at the bottom of each group of detail records.

See the end of this document for a [sample](#) of each format of this report.

## System>Purge>EMV Transaction history

There will probably come a time when you need to purge some of the building EMV transaction history. We don't know of any requirement to retain such information for any specific time (there may be – we just don't know), but we know data builds up over time. This program is actually the same program as the EMV Transaction History report, but it is invoked in a way to enable purging records. Having a separate menu selection will allow you better control over who has the ability to purge this sensitive information.

Being the same program as the EMV Transaction History report, this program has the same parameters plus one more – “5. Print or purge”.

Here is the parameter screen.

```
EMV Transaction history report 7.5.20

Please enter:

1. Store-#           "All"
2. Date range       "Earliest" to "Latest"
3. Transaction type "All"
4. Record type      "All"
5. Report format    Full <EMV tag data>
6. In order by      Store#/date

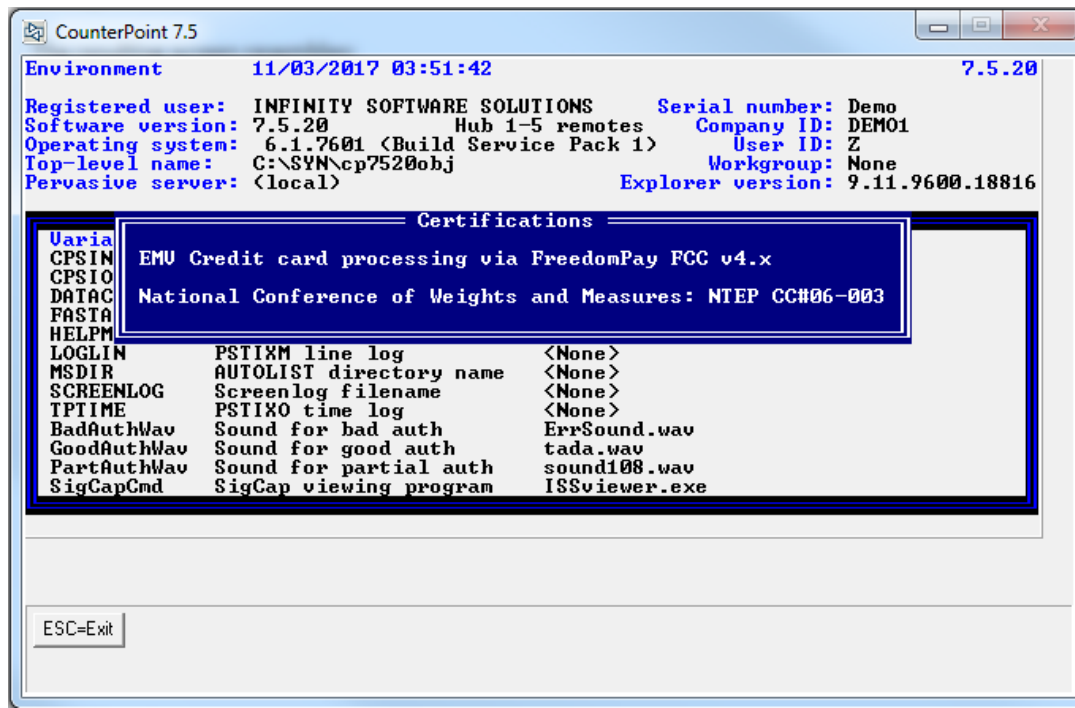
Field number to change ? 
```

See [System>Reports>EMV Transaction history](#) for an explanation of the majority of the parameters. The “Print or purge” parameter allows you to first view the data that you intend to purge before committing yourself to purging it. Note that if you select to purge the data the program will not allow you to display the report to the screen. Also when purging you must select the report format that displays the most information for the selected transaction type. So if you selected “Token” for the transaction type, the “Report format” parameter must be “Brief”. All other values for “Transaction type” require “Full” report format.

## System>Utilities>Environment

This program displays details about CounterPoint's installation and the machine it is running on. It also displays the most commonly used environment variables used to control various aspects of CounterPoint's operation. A client asked us to add a particular environment variable (MSDIR) to the "Advanced" screen of this program. While performing this modification we discovered that pressing <F5> displays the certifications CounterPoint has been awarded. At that time there was only one. We were awarded a certification from FreedomPay for our EMV credit card processing enhancement so we added that in there as well. (The client had already been running CP-EMV for almost a year at that point.)

The resulting screen resembles:

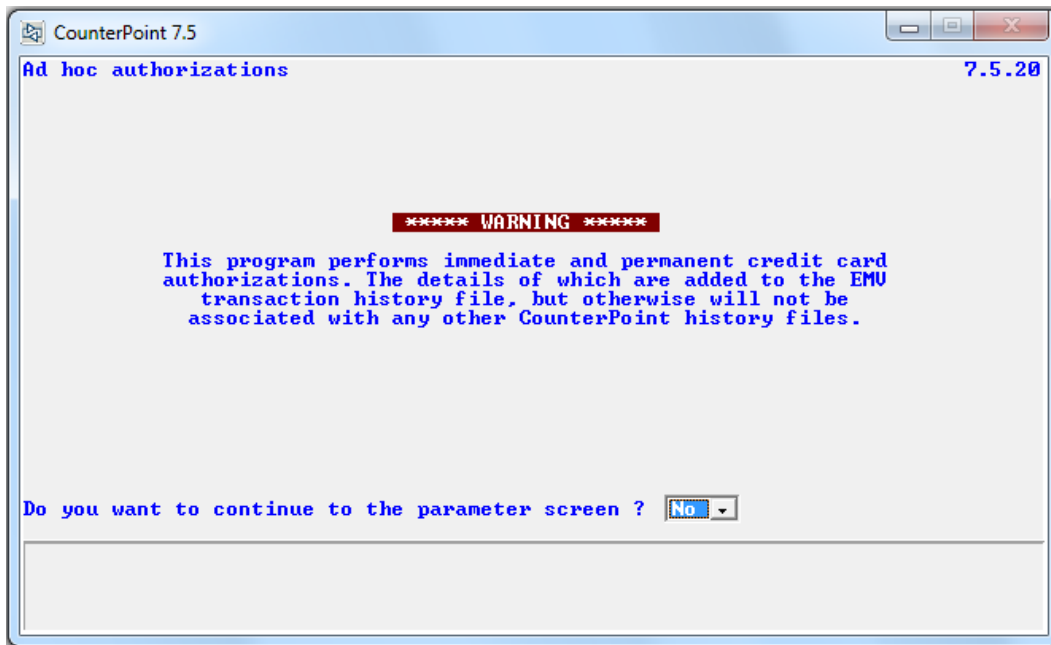


In total, we added 5 new environment variables to the "Advanced settings" screen.

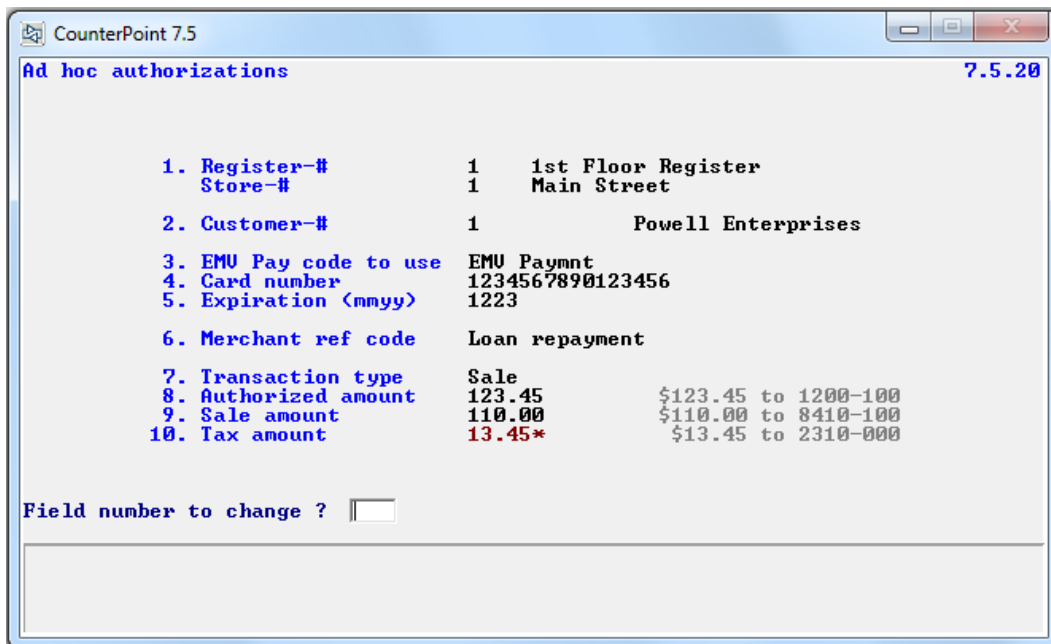
MSDIR	AUTOLIST directory name
BadAuthWav	Sound for bad auth
GoodAuthWav	Sound for good auth
PartAuthWav	Sound for partial auth
SigCapCmd	SigCap viewing program

## System>Utilities>Ad hoc authorizations

This program allows processing a credit card in an ad-hoc manner. You can run a sales transaction, a return transaction or void a transaction from the EMV transaction history file. When you run an authorization via this program there is no CounterPoint history created other than a new EMV transaction history record and distributions. It could come in handy in the rare instance that you need to run a credit card transaction outside of CounterPoint's normal processes. Obviously you should probably restrict access to this program to select people in each store. The operator is first presented a warning screen.



Answering "Yes" takes you to the parameter screen. (We've entered sample parameters in the example below.)



Let's step through the parameters.

**Register-#** This field is required if you intend to employ an Ingenico to perform the authorization. As described multiple times in this document, it is the register number that associates a terminal performing an authorization to the Ingenico that services that terminal. However entering a register number doesn't force you to using an Ingenico. Providing a register number also indicates the store number and so the merchant credentials used during the authorization.

Pressing <F5> at this field will open a new EMV transaction history lookup program. (See [here](#) for details on how that works.) This lookup opens many possibilities. Let's say you need to run an authorization for a customer, but you don't have their credit card information, but you have used their card in a prior authorization. One of the fields returned with all successful authorizations is a token that represents the card used. Each EMV transaction history record has such a token stored in it. Using <F5> to select an EMV transaction record to use as the basis of the current transaction will populate most of the fields on the screen, but these default values can be changed.

**Store-#** If you provide a register number or select an EMV transaction history record to "seed" the current transaction, this field will be populated for you. If you skip entering the register number field this field is prompted for and is required. It is this store that will provide the merchant credentials for the current authorization.

**EMV Pay code to use** This field offers a drop-down list of the EMV pay codes that have been defined for the store. If the customer has at least one EMV customer card available <F7> is offered to select one of those. If you select a customer card the card's token is used for the authorization so no Ingenico is required. The pay code is not stored in the EMV transaction history record so this field is always prompted for.

**Card number** If you selected an EMV transaction history record from the register number field all of the card-related fields are skipped over and the expiration date of the EMV record's token is displayed (if available). The idea being there's an issue relating to that specific transaction which should employ the original card to be resolved. But you can change any of the card-related fields directly from FNTC.

If you selected a customer card from the pay code field all of the card-related fields are displayed and their entry is skipped. The customer card provides these values and so you don't need to key them. But again, you can change any of the card-related fields directly from FNTC.



Otherwise you will land at this field. You can either enter the complete card number (and expiration date) or leave this field blank and allow the card to be swiped. As mentioned above, you must have provided a register number to employ an Ingenico.

If you change the value in this field (from FNTC) you will need to key a complete (unmasked) card number as that will be the card number used. The token expiration date is removed from the screen.

The F3-key is offered to clear the card-related information from the screen.

Expiration (mmyy)	This field is only prompted for when you enter the card number. As depicted on the screen it is formatted mmyy with no slashes or dashes.
Merchant ref code	This is an optional field. It has nothing to do with the authorization process and can be left blank. But CP-EMV puts a value in this field in every other area authorizations can be performed to allow the merchant to identify the origin of the transaction. I suggest you do the same, but it's completely up to you. This field is displayed in the list of transactions in FreedomPay's Enterprise Portal.
Transaction type	This field always allows "Sale" and "Return". If you selected a specific EMV transaction history record from the register number field it will also offer "Void". That's because performing a void requires the original transaction's request-id. A request-id is a 32 character value of absolute gibberish. Even I wouldn't attempt to key one so I don't offer the field for entry. We store that value on the EMV record. If you hand key an authorization you cannot enter that request-id and so you cannot void anything.
Authorized amount	The amount to be authorized. Upon entry the program displays the account number the amount will be distributed to. This is the account number assigned to the pay code. It cannot be changed.
Sale amount	This field represents the general purpose of the payment. The distribution falls under "Sales and Returns" and the default account number reflects this. This field and the following tax field combined must equal the Authorized amount field. Any value entered here cannot exceed the Authorized amount field. Changing the field from FNTC will automatically adjust the tax amount field to equal the Authorized amount field. You can enter zeros in this field and enter the entire amount as a tax amount. Pressing the <F5> at this field allows entering a different account number.

Tax amount This field is optional-ish. While it can be left at zeros and an authorization will be successful there are jurisdictions in the world that have regulations that require this field to be populated. I am not the person to tell you if that applies to you. But I made it available just in case you need it.

The default value is the Authorized amount minus the Sales amount. You can enter whatever value you desire (less than the Authorized amount) and the Sales amount will be adjusted automatically.

Once the Sales amount field has been entered the program calculates the tax values (up to three) based on the sales amount and the customer's tax code. If the customer doesn't have a tax code or the code is not on file the program uses POS's default customer tax code instead. No doubt the entered amount (if entered) won't match the calculated value. When this happens the field is displayed in red with an asterisk next to it. It might look a bit alarming, but it's really just to warn you. Then authorization will still work just fine. The calculated tax values and their account numbers are displayed in grey. The calculated distribution values are initially calculated using the Sales amount. Then these values are adjusted linearly to match the tax amount entered.

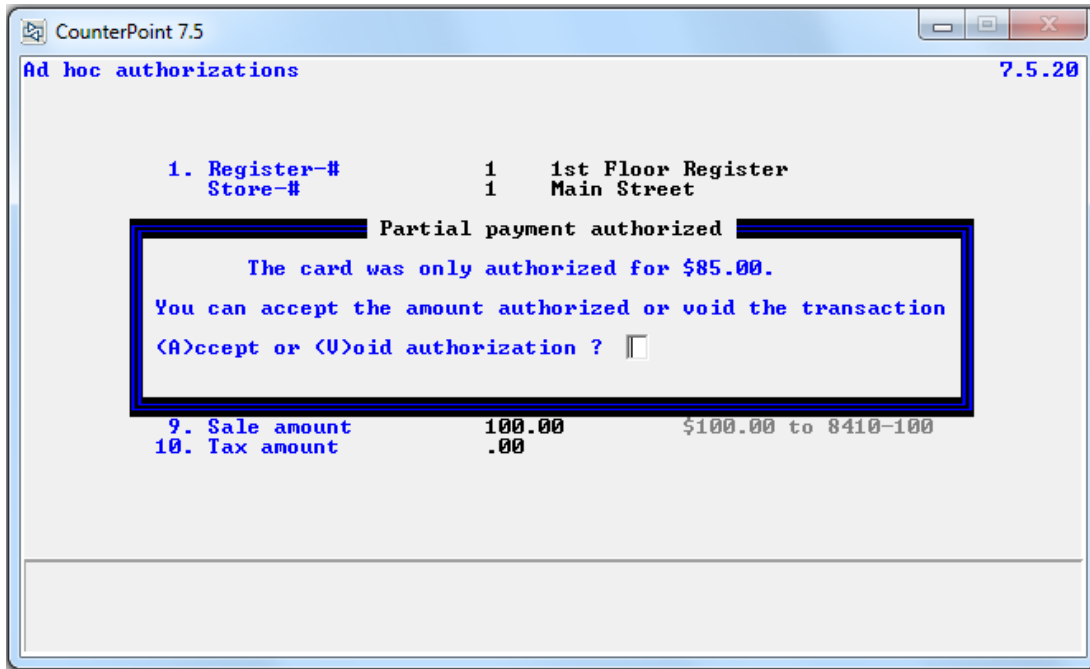
Like the Sales amount field you can press the <F5> key to change the account number for the tax distribution. If you do this the originally calculated distributions are cleared and replaced by the single distribution you key. Note that each time you change the Tax amount field the distributions are recalculated.

Once you've entered the fields and you're at FNTC you can change all of the fields' values (depending on how they were populated). If you selected an EMV transaction history record at the register number field you will not be able to change the customer number nor the card's expiration date directly. The idea is you're addressing a specific EMV transaction for a specific customer. Changing the customer number doesn't make sense. The expiration date field is tied to the credit card number. If there's a change to make to the card information you should start at the card number field.

Pressing <ENTER> at FNTC results in the "last chance" prompt of "Perform the authorization now?". You can answer "Y" or "N". Answering "N" takes you right back to FNTC. If you answer "Y" the program first checks that you haven't left out any required fields. If so, the program provides an appropriate messages and returns to FNTC. Otherwise it continues with the authorization process.

The result of the authorization is displayed on the screen. If the authorization was successful the screen is cleared for the next authorization. If there was a problem with the authorization the screen remains as-is and the program returns to FNTC. This allows you to correct the problem and try again without having to rekey the transaction all over again.

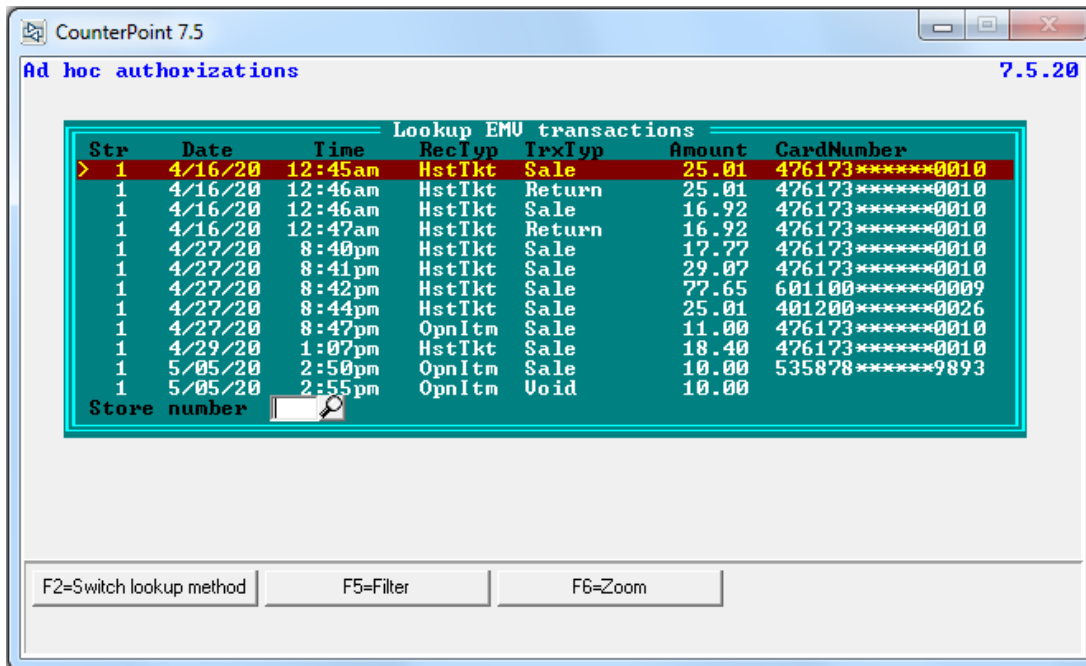
It is possible that the process results in a partial authorization. When this happens the program presents a dialogue asking how you want to handle the situation. In the example below the authorization was for \$100.00, but only \$85.00 was authorized.



If you accept the authorization, the Authorized amount is set to the partial authorization amount and the Sales amount and Tax amounts (and their distributions) are adjusted to reflect the amount actually authorized. If you elect to void the authorization it will not be batched to the processor (as if the transaction never happened) and no distributions will be created. However EMV transaction history will be updated to reflect what actually happened.

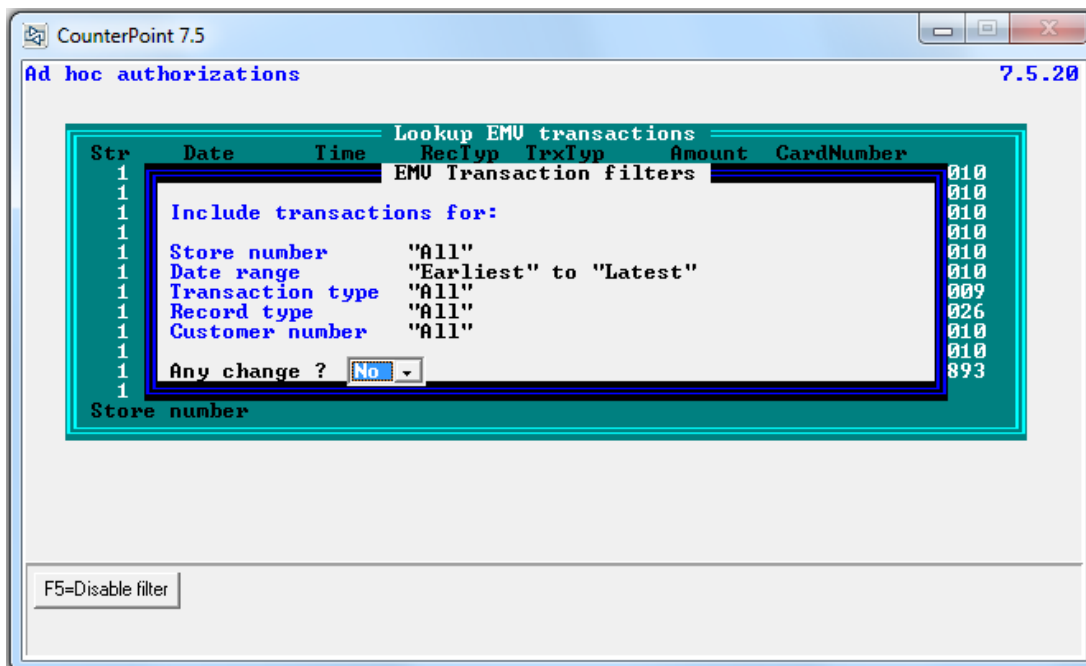
## DataLookup for EMV Transaction File

As mentioned above we created a DataLookup program to allow selecting a specific record from the EMV transaction history file. This program allows searching the file by store/date/time ("Store number") or by date/time ("Date"). Here's an example of the lookup screen.



Pressing <F2> allows switching between the two search methods. By default the other search method is preselected. Changing the search method resorts the transactions and prompts for the new field.

Pressing <F5> allows defining filters to restrict the transactions displayed. The filters are for store-number, date-range, transaction-type, record-type and customer-number. That looks like this.



The store-number and the customer-number fields allow their respective DataLookups. The date-range fields allow specific date values or “Earliest” to “Latest” (or some combination of the two). The transaction-type field is a dropdown selection box of all the transaction types. The selections offered in the record-type filter depend on the transaction type value selected. Answering “N” to the “Any change ?” prompt closes the filter window and redisplay the DataLookup screen using the new filter values.

Pressing <F3> sets the search method to “Store number”, clears all filters and redisplay the screen with the fresh criteria.

Pressing <F6> displays details about the highlighted transaction. The top half of the window displays information to help identify the transaction within CounterPoint. That information is formatted differently for each record type since each record type employs different fields as part of their key. The bottom half of the screen is formatted the same for all record types. Some of these fields are blank or have no meaning for some record types and so are always blank for that record type.

### **Add a Customer On-The-Fly (Point of Sale and Order Entry)**

Point of Sale and Order Entry each provide a program that allows a user to create a new customer record while entering a transaction. We added the new “Allow tokens ?” field to each of these. We also added the ability to press <F7> at FNTC to call the customer card maintenance program. Once there the user has the ability to add customer cards and tokenized credit cards for the customer. We relabeled the field “Card-#/Exp” to “Non-EMV Card-#/Exp” to reflect that the card information entered will not be able to be used in an authorization.

### **File Utilities (All packages)**

#### **Compacting Files**

As you run CounterPoint over the years the files tend to grow – not shrink. Files don’t magically shrink. In fact, the way the Micro Focus file system is designed allows files to grow in size forever – never shrink. When a record is deleted in CounterPoint the physical record is not removed from the file. The record is merely flagged internally as having been deleted. So deleting records in CounterPoint doesn’t shrink the size of the file at all. Having all those deleted records in the file slows the processing of the file. When a program requests the file system to retrieve the next record in a file, the file system will read the next record then look to see if the record has been flagged as deleted. If it has, the file system reads the next record and so on and so on. The computer and hard drive are still spending the time to read these deleted records even if it’s invisible to the user. That is except for the extra time it wastes to skip over the deleted records. That can be noticed.

To correct this Synchronics recommended purging your data (to “delete” undesired records), then exporting and importing the file. Exporting the file creates a text file without any of the deleted records and importing the file creates a new file free of deleted records and with freshly built indices. (Although building an index from an already sorted data file results in a lopsided index structure, but that’s another story.)

There are two problems with that methodology. The first is when a file gets so large that the resulting export file is too large for the operating system to support it. When this happens the file must be exported in chunks the operating system can handle then those chunks must be imported back into a new file. This segues into the second problem – time. Often this process takes so long that it's impossible to accomplish before the start of the next business day. Keep in mind that you cannot export/import data while the POS end-of-day is running. The export/import process would simply be impossible in an environment that's open 24 hours a day.

To solve this conundrum we've added the ability to compact any file using CounterPoint's file utilities. The method is simple. The new code opens two files. The first is the original data file. The second is a new file with the same name and characteristics as the original file but with an "N" at the end of the filename instead of an "F". The program then reads the records from the original file and writes them to the new file. The file system skips over records flagged as deleted so only "live" records go in the new file. At the end of the original file, both files are closed then renamed. The original file is renamed to *filename*.BAK. The new file is assigned the original file's name making it the production file. On a Unix or Linux system, where each logical data file is comprised of two files on the hard drive (one for data and another for the indices) both file are renamed with \*.BAK. This methodology requires only a single pass through the original file – no importing step – and you're done. This is a huge benefit for you and frankly, a lot of tedious code for me.

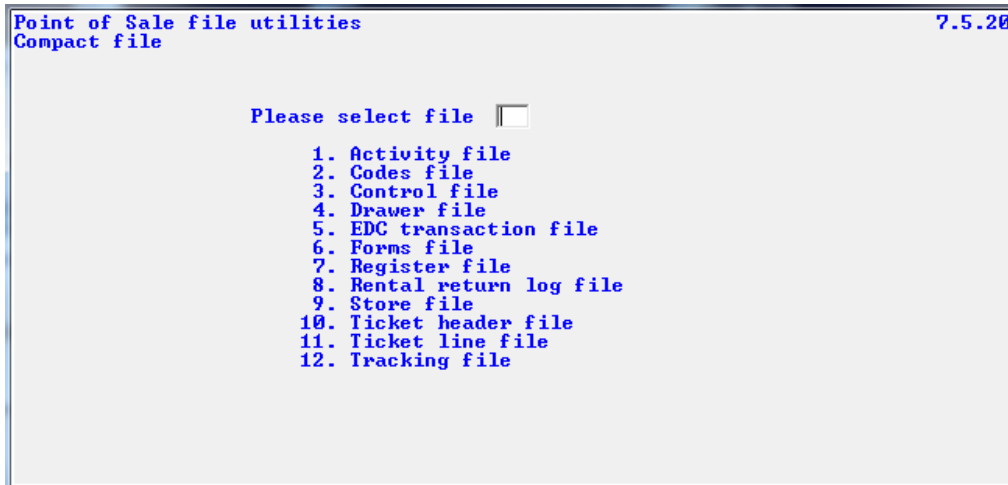
But there is a caveat at the time of this writing. We know certain operating systems will break a CounterPoint file into 2gig segments. While this process will compact these files correctly it fails when it comes to renaming the resulting segments. This is merely because we don't have a sample of such a file to use in our coding and testing. We don't have an actual sample of how the file segments are named. If you have such a situation call us so we can handle the coding. But you can still use the file utilities as they are and finish renaming the segments manually (until we have coded that last part).

So let's take a look at what the compact process looks like. When you run file utilities you'll find a new menu selection "6. Compact files" for each package.

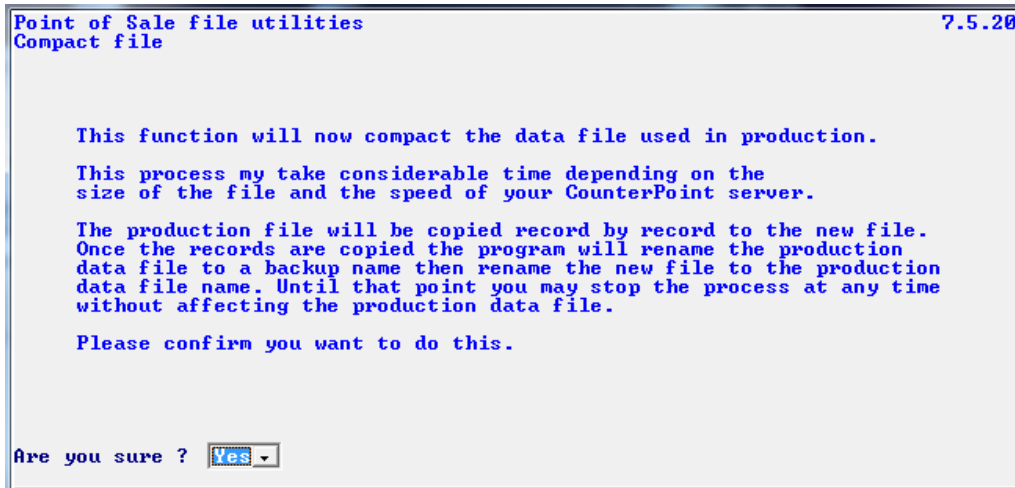
```
Point of Sale file utilities 7.5.20

Please select █
  1. Rebuild a file
  2. Export a file
  3. Restore from an export file
  4. Initialize files
  5. Verify files
  6. Compact files
```

Selecting that displays the usual file selection screen you've already seen many times.



Pick a file and press <ENTER> at FNTP takes you to this new "last chance" screen.



Answer "Yes" and the process begins. The screen displays a progress counter so you can see something is happening. Notice that the counter displays the record count in multiples of ten until it completes. Then it displays the exact record count. Over the years we've discovered that displaying the record count on every record can have a measurable effect on how long the process takes. Apparently console I/O takes significant time on some platforms.

Here's the progress screen.

```
Point of Sale file utilities 7.5.20
Compact file

Compacting DEMO1/PSDATA/PSIRKF.DAT
Record 000000117

Compacting complete - press ESC to return to file selection menu
```

As the program runs it offers you <F1> to pause the process. Once paused, you can choose to continue or abort the process. If you choose to abort, the process ends immediately and neither file is renamed.

### Automatic File Utilities

When you run the automatic file utilities (*File Utilities>Automatic*) you select the Application-ID on the first parameter screen. We didn't make any changes to this screen. On the second parameter screen you select the function you wish to perform. We added "Compact" to the end of that list. It operates in exactly the same manner as described above. Here's the change to that screen.

```
Automatic file utilities 7.5.20

Please enter:
1. Application ID PS
2. Function
3. Filename
Rebuild
Export
Restore
Export/restore
Rebuild/export/restore
Initialize
Optional initialize
Verify
Verify/export
Compact
```

### Exporting Files

Another addition to the file utilities is the ability to export files forward or backward. Normally the file utilities export a file by reading the next record then the next record in a forwards direction starting at the beginning of the file (or whatever value the operator indicated as the starting key). Reading the file from the end of the file towards the beginning allows one to get past a corrupt section of a file while losing as little data as possible.



Previously if a file developed a “bad spot” the operator would need to export the file until the process failed. He’d then need to open the exported data file with an editor to determine the primary key of the last record exported. (That’s no small feat in itself.) He’d then have to guess the primary key value for the next readable record and try to export the remainder of the file.

Having the ability to export the file backwards greatly simplifies this process. One could simply export the file (creating a new export file) forwards from beginning to end until the process dies. That results in an export file with the first “half” of the retrievable data. Then export the file backwards (appending to the existing export file) from the end of the file to the beginning until it dies. That appends the last “half” of the retrievable data to the export file. Now import the resulting export file to a new data file and you’re done ! Of course that assumes there’s only a single bad spot in the file.

Determining the correct values to enter for the starting and ending key parameters is impossible unless you know where to find a description of each file’s primary key structure. It would be nice if CounterPoint would tell you where to find this information and how to use it right ? Well, we did and you’re welcome ! We changed the export a file parameter screen to include this information as well as the new Forwards/Backwards parameter.

```
Point of Sale file utilities 7.5.20
Export a file

Creating new export file: DEMO1/EXPORT/PSTRKF.EXP

1. Read records forwards or backwards ? Forwards
2. Starting file key Backwards
3. Ending file key

To find the composition of the primary key for the file being exported
look in the \TECH\DOSCOPY directory of your CounterPoint installation CD.
Look for a *.FDX file that matches the 6-character name of the file being
exported. Generally speaking the name of the primary key will be PP-XXX-KEY
where "PP" is the package-name, "XXX" is the file designator and "-KEY"
denotes the group field of the file's primary key. Keep in mind that some
files carry several types of records with different primary key structures.
```

So let’s look in that \TECH\DOSCOPY directory and how to use what’s in there. There’s a lot – over 1,200 files. All of these files are called copyfiles. They are little chunks of COBOL source code that can be copied into a program at compile time. This allows programmers to define a file (or a piece of code) once and reuse that code in any number of programs without having to key the same stuff over and over. If a change needs to be made to the contents of a copyfile it is much simpler to make the change in the one copyfile then recompile all of the programs that use the copyfile than to make the same change to the source code of every program that would otherwise have that code in them. (Thank you Grace Hopper !)

Every data file in CounterPoint has a \*.DS, an \*.FD, an \*.SL and a \*.WS copyfile. Master information files and transaction files also have an \*.FDW copyfile. Exportable files have an \*.FDX as well. These last files are the ones that describe the export file and so are the copyfiles we want to use.

The naming convention used is rather cryptic, but simple. The first two characters are the application-id such "AP" for "Accounting", "AR" for "Customers", "IM" for "Inventory" and "PS" for "Point of sale". Ummm... That doesn't actually look right does it ? It makes sense if you consider CounterPoint's roots. CounterPoint is an evolution of a long-defunct, but popular back-in-the-day RealWorld Accounting System. RealWorld's packages were named "Accounts Payable" (A/P), "Accounts Receivable" (A/R) and so on so these two-character package designations made more sense in that context. We actually played a part in the naming convention confusion when we designed CounterPoint's Basic Accounting application. Having been raised in the RealWorld coding culture we continued the same naming convention in Basic Accounting which resulted in four different package designations ("AP", "BA", "CR" and "GL") being used to identify files for a single Basic Accounting package. Sorry 'bout that.

As penance I'll list out the package designations and the application-id they're associated with.

<u>Prefix</u>	<u>CounterPoint</u>	<u>RealWorld</u>
AP	Accounting	Accounts Payable
AR	Customers	Accounts Receivable
BA	Accounting (short for "Basic Accounting")	
CL	Custom labels	
CR	Accounting	Check Reconciliation
DI	Data Interchange	
EC	eCommerce	
GL	Accounting	General Ledger
IM	Inventory	Inventory Management
MS	MultiSite	
OB	Open to Buy	
OE	Order Entry	Order Entry
PA	Purchasing	Purchase Orders
PS	Point of Sale	
SA	Sales History	Sales Analysis
SY	System	System Manager

So the first two characters are the application-id. The next three characters are the file descriptor. That's the meat of the filename since the last character of the filename is always an "F". Even in these three short characters there is a convention. If a file is a transaction file that is posted the third character is an "X". So "ARCTXF" is the Cash Receipt transaction file. "IMITXF" is the Inventory transaction file. Files that carry code-type records are named with "COD" in them like "ARCODF", "IMCODF" and "PSCODF". Otherwise they assign values that look right like the customer file "ARCUSF", the item file "IMITMF" or the vendor file "IMVENF".

The best way to get a file's "on-disk" filename is to let the file utilities give it to you. Go into the package's file utility menu selection as if you want to export a file. On the file selection screen, press <F2> for "Display filenames". There they are (for that package).

```

Customer file utilities                                     7.5.20
Export a file

Please select file

1. Adjustments trx....ARATXF
2. Cash receipts trx..ARCTXF
3. Closed item.....ARCLSF
4. Codes.....ARCOF
5. Control.....ARCTLF
6. Customer.....ARCUSF
7. Finance charge trx.ARFIXF
8. Forms.....ARFRMF
9. Open item.....AROPNF
10. Renumber customers.FIXCUS
11. Ship-to address....ARSHPF

```

Notice file number ten ? It's named "FIXCUS" even though it's clearly a Customers file. What good is a convention if you don't break it now and then right ? And Synchronics did – again and again.

<u>File</u>	<u>Description</u>	<u>Package</u>
FIXCUS	Renumber customers file	Customers
FIXGRD	Change grids file	Inventory (not in file utilities)
FIXITM	Renumber items file	Inventory
FIXUSR	Renumber users file	System
FIXVEN	Renumber vendors file	Inventory

Write down the name of the file you're interested in, press <ENTER> to close the filenames screen then proceed to the "Export a file" parameter screen.

Now back to figuring out the composition of a file's primary key. In the \TECH\DOSCOPY directory you'll find that each exportable file has an \*.FDX file that describes it. Every file has a \*.WS file as well, but you won't need that unless the file contains different types of records. (We'll touch on that later.)

Let's pick an easy one first. Open ARCUSF.FDX with your favorite editor (or just NotePad). All of the keys for a file are always defined at the top of the file. You're looking for a field that ends with "-KEY.", but not "-ALT-KEY.". The "-KEY" field is the primary key. It will be named like "EXP-fff-KEY." where "fff" is the file designator name. So the primary "key" for the exported customer file is "EXP-CUST-KEY". Note the level number to the left of the name. In this case it is "05". The primary key is a group field that is always comprised of one or more elementary fields. Basically any field below the primary key declaration with a numerically higher level number is part of the primary key until you encounter another field with the same level number (or lower) as the primary key. That sounds complicated, but visually it's quite obvious. It helps that each numerically higher level field is indented a little more to the right.

Let's compare the primary key definitions for two files. Below is the file definition for the exported customer file (ARCUSF.FDX).

```
01 EXP-CUST-REC REDEFINES TEXT-REC.
03 EXP-CUST-ALT-KEY.
    05 EXP-CUST-NAM-UPPER          PIC X(25).
    05 EXP-CUST-KEY.
        07 EXP-CUST-NO              PIC X(12).

03 EXP-CUST-CHNGD-REC-ALT-KEY.
    05 EXP-CUST-CHNG-REQST-TYP     PIC X(1).
    05 EXP-CUST-NO-ALT             PIC X(12).
```

Here's the primary key for the POS header file (PSHDRF.FDX).

```
01 EXP-HDR-REC REDEFINES TEXT-REC.
03 EXP-HDR-ALT-KEY-1.
    05 EXP-HDR-TYP-ALT-1           PIC X(1).
    05 EXP-HDR-ALT-KEY-DATA-UPPER  PIC X(25).
    05 EXP-HDR-KEY.
        07 EXP-HDR-TYP             PIC X(1).
        07 EXP-HDR-REG-NO          PIC 9(3).
        07 EXP-HDR-TICKET-NO       PIC 9(6).

03 EXP-HDR-CHNGD-REC-ALT-KEY.
    05 EXP-HDR-CHNG-REQST-TYP     PIC X(1).
    05 EXP-HDR-TYP-ALT            PIC X(1).
    05 EXP-HDR-REG-NO-ALT         PIC 9(3).
    05 EXP-HDR-TICKET-NO-ALT      PIC 9(6).
```

The primary key for the customer file is the single field "EXP-CUST-NO". Looking to the right you can see "PIC X(12)". "PIC" is short for "Picture" and will occur on every elementary field. The character that follows will be either an "X" for alphanumeric or a "9" indicating a digit. The number in parenthesis tells you how many characters (or digits) the field is. So "PIC X(12)" indicates the customer number is an alphanumeric field that is 12 characters in length. If there's no parenthesized number then the field is one character (or digit) long. So when you're filling the starting and ending keys while exporting the customer file, you'll be keying a value that is no more than 12 characters in length. Anything beyond 12 characters will be ignored.

Which segues to another important topic when keying the starting and ending key fields – justification. Master fields such as customer-number, item-number, vendor-number and such need to sort correctly in reports and lookups. When a numeric value is stored in one of these types of fields it is right-justified and zero-filled (RJZF). That means the field's contents are shifted all the way to the right and the blank spaces that are created on the left side are filled with zeros. Nonnumeric values are stored exactly how they were keyed (which can be bad). If numeric fields weren't RJZF their primary keys would sort incorrectly like this.

1  
10  
11  
2  
21  
3

Applying the RJZF rule to a numeric field normalizes the field in numeric form and results in a field that sorts correctly.

```
000000000001
000000000002
000000000003
000000000010
000000000011
000000000021
```

Keep this in mind whenever you're figuring the starting and ending keys for files whose primary key includes a "master" field. So if you use numbers for your customer numbers and you want to export the first 10,000 customers from your customer file you'd use the starting and ending keys "000000000001" through "000000010000" not "1" through "10000".

Now let's look at the more complicated primary key structure from the POS header record (from the example above). That key is comprised of three fields – TYP (one character), REG-NO (3 digits) and TICKET-NO (6 digits). The REG-NO and TICKET-NO fields are pretty straightforward numeric fields. Since they're defined as numeric they're automatically RJZF upon entry. The field TYP is a single alphanumeric character. Theoretically the field TYP could contain any character a person could type, but in reality it will only have a handful of possible values. We call these types of fields "magic fields". That's not because they are endowed with special powers. It merely means they contain predefined values with special meanings that determine how the record is handled. Generally they are hated by programmers because it can be difficult to remember the permissible values, but CounterPoint provides a way to manage them.

To find the allowable values for this magical field we need to refer to the file's \*.WS copyfile. (I told you we'd be getting back to this.) Open the file PSHDRF.WS with your favorite editor. There you'll find the lines:

```
01 PS-HDR-FIL-NAM                PIC X(50) VALUE
                                "PSDATA/PSHDRF".

01 PS-HDR-FIL-STAT                PIC X(2) .
  88 PS-HDR-FND                    VALUE "00".
  88 PS-HDR-NOT-FND                VALUE "23".

01 PS-HDR-FIL-OPN-STAT            PIC 9(1) VALUE ZERO.
  88 PS-HDR-FIL-IS-OPN            VALUE 1.

01 PS-HDR-TIX-TYPS.
  03 PS-HDR-TYP-IS-TIX            PIC X(1) VALUE "T".
  03 PS-HDR-TYP-IS-DRW            PIC X(1) VALUE "D".
  03 PS-HDR-TYP-IS-HLD            PIC X(1) VALUE "H".
  03 PS-HDR-TYP-IS-LAY            PIC X(1) VALUE "L".
  03 PS-HDR-TYP-IS-NEW-LAY        PIC X(1) VALUE "Y".
  03 PS-HDR-TYP-IS-ORD            PIC X(1) VALUE "O".
  03 PS-HDR-TYP-IS-NEW-ORD        PIC X(1) VALUE "N".
  03 PS-HDR-TYP-IS-QUT            PIC X(1) VALUE "Q".
```

The first several fields carry the file's default filename, the file status and a numeric flag field that indicates if the file is currently open. These are the minimum lines such a file will carry. We're interested in what is below these lines. If a file has any magical fields the allowable values for these fields are usually enumerated there. You need to keep these magical fields in mind when keying your starting and ending keys. Looking at the list of values you should see that the TYP field in PSHDRF will never have an "A", "B" or "C" character in it. So the key range "A" through "C" wouldn't find any record to export. In fact the range "A" through "D" wouldn't find anything either even though "D" is a valid TYP field value. That's because the primary key for PSHDRF is comprised of three fields. Being numeric, it is impossible for the second and third fields to be blank. Therefore the minimum primary key for the first "D" type record in PSHDRF will be the value "D000000000". That's an impossible value since neither REG-NO or TICKET-NO can blank, but still "D000000000" is programmatically greater than "D" so the first "D" record would not be exported. But "D" would be a legitimate starting key value for retrieving all of your "D" type records. Get how that works ?

Well I think I've thrashed on this topic enough. It's far more explanation than Synchronics ever gave anyone on how to export a file.

### Handling Locked Records While Exporting a File

We also addressed a problem that has plagued the CounterPoint file utilities forever. When exporting a file if the program encounters a record that is locked it loops on that record until the record is unlocked. While looping it writes out a copy of the locked record over and over again. And neither operator has any indication this is happening.

We added code that checks for a record locked condition each time it reads a record. If a locked record is encountered the program displays a message on the screen and tries again. It will sit in this loop until the record is unlocked, but the difference is it's not writing thousands of copies of the same record to the export file and it's telling the operator what is happening. Here's what the message looks like.

```
Point of Sale file utilities 7.5.20
Export a file

Creating new export file: DEMO1/EXPORT/PSREGF.EXP

1. Read records forwards or backwards ? Forwards
2. Starting file key "First"
3. Ending file key "Last"

<Stuck on locked record>

To find the composition of the primary key for the file being exported
look in the \IECH\DOSCOPY directory of your CounterPoint installation CD.
Look for a *.FDX file that matches the 6-character name of the file being
exported. Generally speaking the name of the primary key will be PP-XXX-KEY
where "PP" is the package-name, "XXX" is the file designator and "-KEY"
denotes the group field of the file's primary key. Keep in mind that some
files carry several types of records with different primary key structures.
```

Once the record is unlocked the program writes the record to the export file, removes the message from the screen and continues on to the next record.

## File Utilities>Customers

We added fields to several Customer data files. So we had to modify the Customers file utilities to support these new fields. We added the new field to the export, import and data validation routines.

## File Utilities>Order Entry

We added fields to several Order Entry data files. So we had to modify the Order Entry file utilities to support these new fields. We added the new field to the export, import and data validation routines.

## File Utilities>Point of Sale

We added fields to several Point of Sale data files. So we had to modify the Point of Sale file utilities to support these new fields. We added the new field to the export, import and data validation routines.

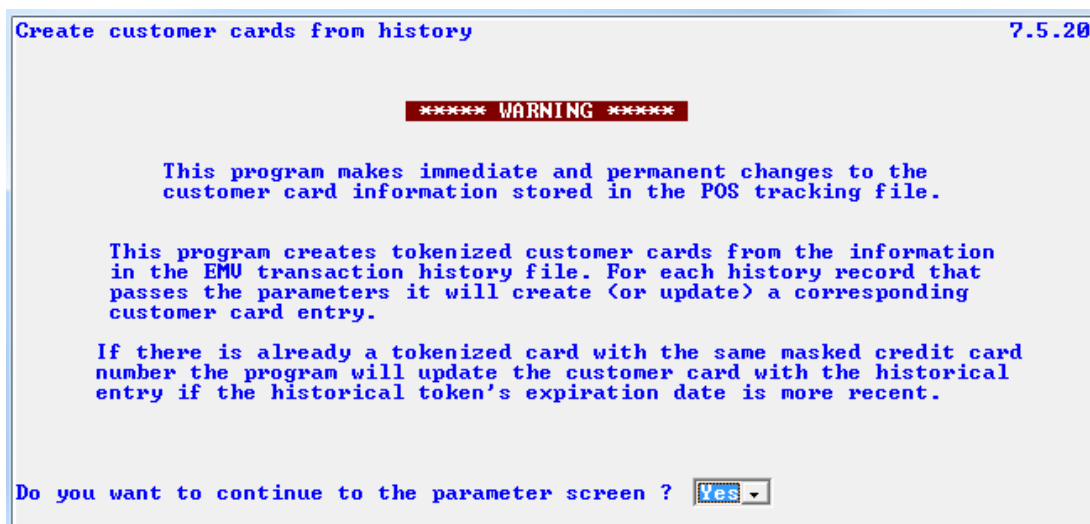
## File Utilities>System

We added fields to several System data files. So we had to modify the System file utilities to support these new fields. We added the new field to the export, import and data validation routines.

## File Utilities>Special>Customers>Create tokens from history

If you've been using CP-EMV for any length of time you should have a decently populated EMV Transaction History file. This file carries the information related to every successful EMV transaction since you started using CP-EMV. In that file are enough data to create EMV-style customer cards. That's what this program does.

Running this program will present you an initial splash screen.



Answering “Yes” takes you to the parameter screen.

```
Create customer cards from history 7.5.20

Please enter:

1. Transaction date range 1/01/20 to "Latest"
2. Customer-# range "First" to "Last"
3. Customer type "All"
4. Customer category "All"
5. EMV Pay code to use EMU EMU Paymnt

Field number to change ? 
```

Parameter “**1. Transaction date range**” allows you to break the processing up by the posting date of the history. While this process is far faster than tokenizing customer cards (because the tokens already exist – they’re just in the history file) you might have a lot of history and you might only want to include customers that have bought something from you in the past year. Note that the starting date offers <F1> for a date that is two years prior to the current system date. That’s because token expire in two years. So it wouldn’t make any sense to create customer cards for already expired tokens. Use a reasonable value.

The next three parameters “**2. Customer-# range**”, “**3. Customer type**”, and “**4. Customer category**” are self-explanatory and allow you to filter the customers processed based on customer-number, type or category. We’ll leave it to you to figure if they make any sense for you.

Parameter “**5. EMV Pay code to use**” has the exact same purpose as in the “Tokenize customer cards” program described in the prior section. Just enter the EMV pay code you want to use. If you only have one defined in your system the program will default to it and you can’t change it.

As the program runs (starting at the oldest history towards the newest) it skips over records for customers that don’t match the customer filters and records for the default customer-# (from the store configuration record). It then loads the customer’s existing customer card entries. It looks to see if the masked card number already exists in the list of cards. If it does and the token expiration date is newer than the existing entry it replaces the existing entry with the new info. If the card number isn’t in the list of customer cards the program determines the next available slot for the cards and creates the new entry.

As the program reads through history it’s probably going to encounter records for the same customer and possibly the same credit card over and over (hopefully). So it will update the same customer card entry several times. So at the end of the report when it shows a total number of cards updated that seems too high, this is the reason.

There’s a sample of the resulting [report](#) at the end of this document.



## File Utilities>Special>Customers>Set customer allow tokens

If you have set up CP-EMV to automatically save token at the customer level this program will be very important to you. You'll need to set the new "Tokens?" field on each and every customer record that will utilize the feature. If you have hundreds of customers that need this flag set it could be quite a task to do this manually. We wrote this program to (hopefully) make this process much simpler.

Opening the program presents the parameter screen.

```
Set customer allow tokens                                     7.5.20

Please enter:

  1. Enable/disable tokens  Enable
  2. Customer-# range      "First"   to "Last"
  3. Customer type         "All"
  4. Customer category     "All"
  5. Report or update ?    Update

Field number to change ? 
```

The purpose of the parameters ought to be pretty apparent, but we'll go through them anyway.

**"1. Enable/disable tokens"** allows "Enable" and "Disable". At first you might think the parameter is pointless since the whole point of the program is to turn the token flags on. But sometimes it is quite useful to use negative logic to arrive at the desired outcome. Let's say all of your customers have been assigned one of ten categories and you want nine of them to automatically save their tokens. Without this parameter you'd need to run this program nine times – once for each category. To be thorough you might also want to manually set the flag for the customers for the one category that won't save tokens. And what if there were customers on file that somehow slipped through the cracks and weren't assigned a category at all? Having this parameter allows you to handle all of your customers in merely two passes. Your first run would enable the "Token?" field for all customers. The second run would disable tokens for the customers for the one category.

The **"Customer-# range"**, **"Customer type"** and **"Customer category"** parameters are self-explanatory. You don't need an explanation for these.

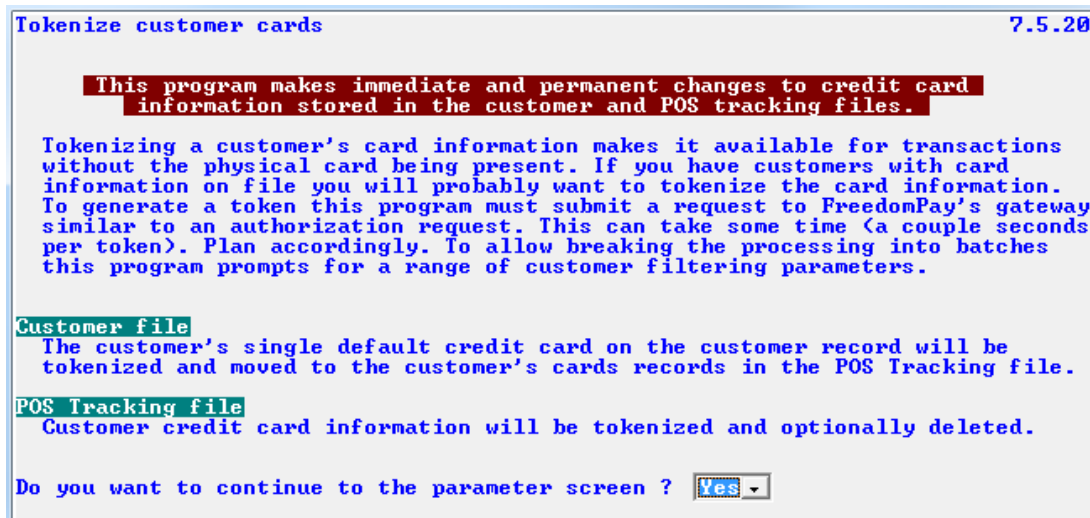
The parameter **"5. Report or update?"** allows the selections "Report-only" or "Update". Despite the literals both values create a report. "Report-only" gives you the ability to create a report of what would happen if you were to run the program with the current parameters, but no records are changed. In this mode you can display the report to the screen. But if you run the report in update mode the program forces you to send the report to a printer or disk. That's to enforce an audit trail since data will be changed.

## File Utilities>Special>Customers>Tokenize customer cards

No doubt after employing CP-Gateway for some time your system will have customer credit card data throughout several files. Most of this data is encrypted (but not all – which is very bad). But part of the reason to use CP-EMV is to take CounterPoint out of PCI-compliance scope. Having credit card data (encrypted or not) in your data files keeps CounterPoint in scope. To correct that you need to mask or delete all credit card numbers in your data files.

You might have “live” credit card information on file in order to bill customers without their card being present. You don’t have to give up that convenience when using CP-EMV. However you will need to convert the credit card information to tokens to be out of scope. Tokenizing is a process that stores the credit card information elsewhere (in FreedomPay’s database) and creates a token that represents that stored card data on FreedomPay’s system. When submitting an authorization request, CounterPoint passes the token along with the transaction data. FreedomPay replaces the token with the actual credit card information as it communicates with the processor. Thus CounterPoint can store customer card data (in the form of the card’s token) and perform billing without the physical card being present.

To accomplish both of these tasks we created a new program named “Tokenize customer cards”. When the program first starts it displays a “splash” screen to warn and inform the operator what is about to happen.



Answering “No” returns you to the menus. Answering “Yes” takes you to the parameter screen displayed below.

The parameter screen is then displayed with the parameters below. Note that the program optionally saves its parameters so you can always save where you’re at in the process and pick it up later.

```

Tokenize customer cards 7.5.20

Please enter:

1. Customer-# range          "First"      to "Last"
2. Customer type             "All"
3. Customer category         "All"
4. EMU Pay code to use       EMU  EMU Paymnt
5. Delete tokenized cards ?  Y
6. Mask printed card numbers ? Y

This program creates a report of the records it changes. When creating tokens it
can take a while to complete. During this time the selected print device is
locked. For this reason we suggest sending the report to "Print to disk".

Field number to change ? 

```

Parameters “**1. Customer-# range**”, “**2. Customer type**” and “**3. Customer category**” are self-explanatory. They allow processing the customer card data in chunks. You may have thousands of customer records each with several credit cards on file. Tokenizing a card may take a couple of seconds (usually less than a second). So the process may take a while. This parameter will allow you the opportunity to test how long the process takes then run the program in appropriately-sized ranges of customer numbers (or types or categories).

Parameter “**4. EMV Pay code to use**” might be a bit confusing, but it serves an important purpose. When you define a customer card one of the fields that is prompted for is a pay code to be associated with the card. This program cannot reuse the pay code already assigned to the CP-Gateway-style cards it tokenizes because EMV credit card pay codes don’t have any prompt fields. Regular credit card pay codes will have these prompts for the card number and expiration date. So the program needs to assign a different pay code – an EMV pay code – during the tokenizing process. It tries to determine which pay code to use automatically. When the program starts it reads through the pay codes defined for the current store looking at the pay code’s type (credit/debit card) and the prompt fields. It counts the number of credit/debit pay codes with no prompts. Those will be EMV pay codes. If none have been defined the program will not allow tokenizing card data since it cannot assign an appropriate EMV pay code. If this is the case you’ll receive the warning message “No EMV customer card has been defined” when the program first starts. To get past this situation you need to go to *Setup>Customers>Control>(Customer Cards)* and add an EMV pay code entry to field “2. Customer card types”.

If there is a single EMV pay code defined for customer cards the program will default parameter 4 to that entry. If more than one is defined the program will allow you to select the one to use. Since CounterPoint doesn’t provide an <F9> lookup for customer cards we coded the program to offer a selection window that offers a selection of the EMV pay codes entered into the customer cards setup. Select the one to be assigned to the new tokenized customer card records.

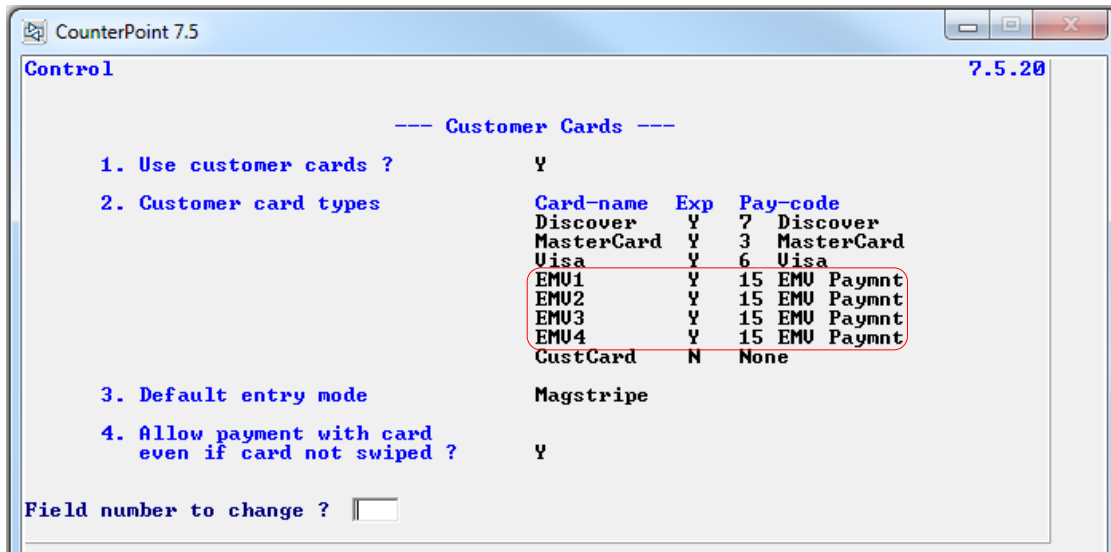
Parameter **“5. Delete tokenized cards ?”** allows you to delete the credit cards that get tokenized as they are processed. You might think one might always want to delete the non-EMV customer cards, but if you have several stores and not all of them have transitioned to CP-EMV that would make the old format cards unavailable to the stores that are still using CP-Gateway. (Silly people !) It’s pretty straightforward. Either way, any existing card information is removed from the customer record. This is because there isn’t enough room in the customer record to store the tokenized card’s information. The program moves the card information from the customer record to the POS tracking file. Thus the card will disappear from the customer maintenance screen and appear as the first EMV-compatible card in the “Customer cards” window.

Parameter **“6. Mask printed card numbers ?”** controls whether you want to mask the credit card numbers in the resulting report. As a rule (with ISS) when we write a program that changes data we create a report to show the before and after results. This gives an audit trail of what happened. But this program deals with credit card numbers which are quite sensitive in nature. The report will allow you to print the full, unencrypted credit card number (as they appear in CounterPoint). If you don’t mask the credit card numbers the resulting report becomes a liability. It would be a great source of information if something were to go wrong or you wanted to archive the information. But you can’t merely run the report to “Print to disk” and leave it there for all of your employees to access. That’s OK if you mask the credit card numbers. If you don’t mask them you should “Print to disk” then physically print the report then store the report somewhere inaccessible.

There’s a caveat concerning selecting the pay code to use. CounterPoint allows up to eight cards for a customer. Typically you would define a separate customer card entry for each type of credit card you accept such as “Discover”, “MasterCard”, “Visa”, “Amex” and so forth. You’d then assign each entry with the pay code that is associated with that card type. So technically you could have up to eight cards for any one customer each with a different pay code. But CP-EMV can’t differentiate between card types like that. That’s because the clerk is completely separated from handling the customer’s card. The clerk merely selects an EMV pay code (you’ll probably only define a single such pay code anyway) and the customer swipes/inserts/taps whatever type/brand of card they want to use.

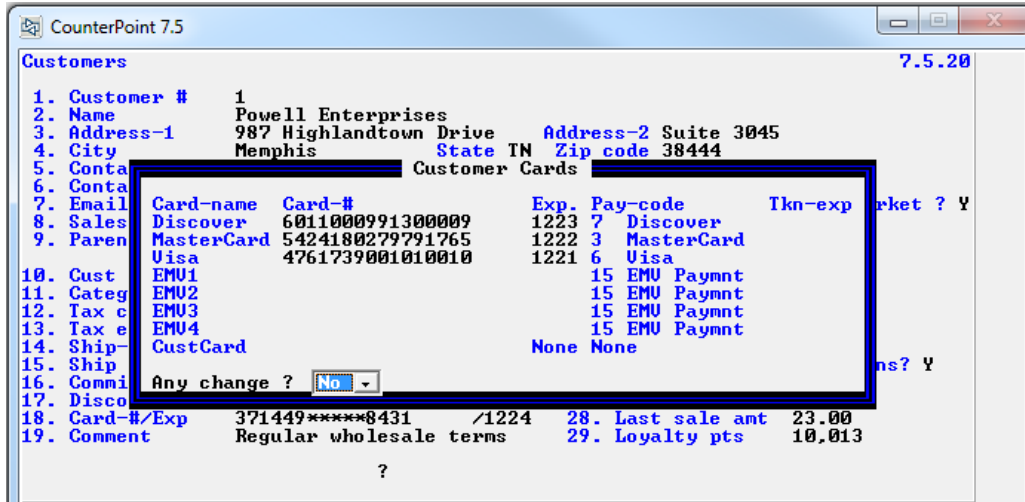
CounterPoint ties the customer card type to the pay code by its position in the list of defined customer card types. So each customer card is assigned a number that ties the card to a specific entry in the list of customer card definitions. If this program were to merely tokenize each existing customer card and assign each to the entered pay code you would end up with several tokenized cards all assigned to the same customer card table entry. That won’t work. Only the first card assigned to the entry would ever be displayed. The rest would be forever hidden – inaccessible even to delete.

So this program expects you to define several EMV card type entries in *Setup>Customers>Control>“Customer Cards”* to carry the various credit cards your customers may already have on file. You can add these new entries at the end of the existing entries like this:

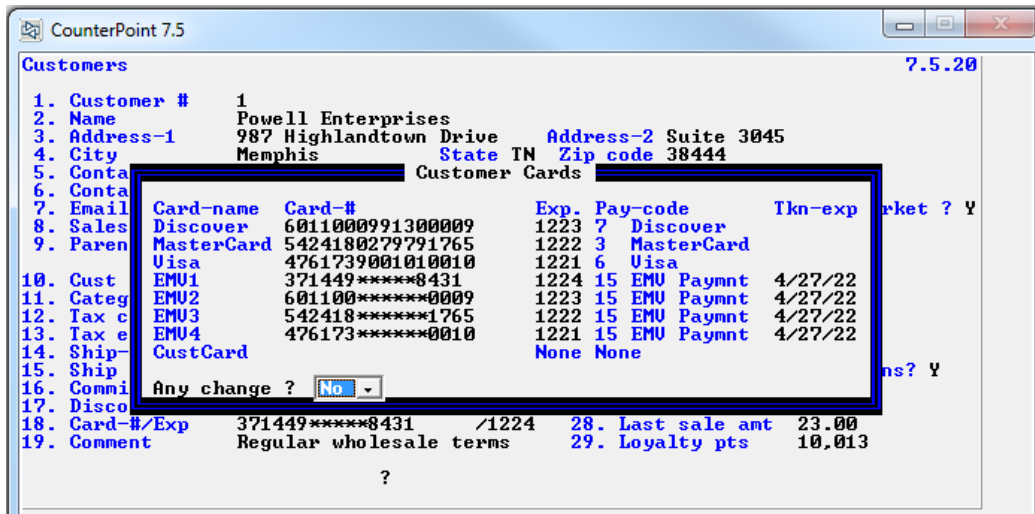


If you define several EMV card types like this the parameter “4. EMV Pay code to use” will open a selection box displaying each one. Select any of them – it doesn’t matter which one if they are all associated with the same pay code. This example numerates each EMV entry as “EMV1”, “EMV2” and so on, but that’s not required. You could assign all of them the exact same label such as just “EMV”. When you select a value to use the program is retaining the pay code associated with the selection – not the specific customer card entry you selected. As the program runs it tokenizes a card then assigns it to the next available EMV type customer card entry.

So given a customer’s cards window looks like this before running the program...



It will look like this after running the program.



Note that when I ran this test case I answered “No” to “5. Delete tokenized cards ?” so you can see how the original cards are left in place. In the last screenshot you can see the credit card on the customer record (hiding behind the “Customer Cards” window) has been assigned the first available EMV slot. Then each of the existing cards was assigned the next open EMV slot as each was processed. The EMV entries each have a masked credit card number and a token expiration date.

Note that if you define fewer EMV cards than you already have non-EMV cards you run the risk of losing customer card data since there won’t be enough EMV entries to store all of a customer’s card data. (If you have a customer that actually has that many cards on file.) If that situation exists you’ll receive the warning message “Not enough EMV customer cards defined” when the program first starts. You can still continue to run the program. It’s just a warning that you might lose some customer card data.

### **File Utilities>Special>System>Create a new company**

When you run the menu selection to create a new company one of the things the program does is build the directory structure to support the new company. We modified this program to create the new *company-id*/SYDATA/IMAGES directory for the signature capture files and the *company-id*\OTE directory to support Offline Ticket Entry.

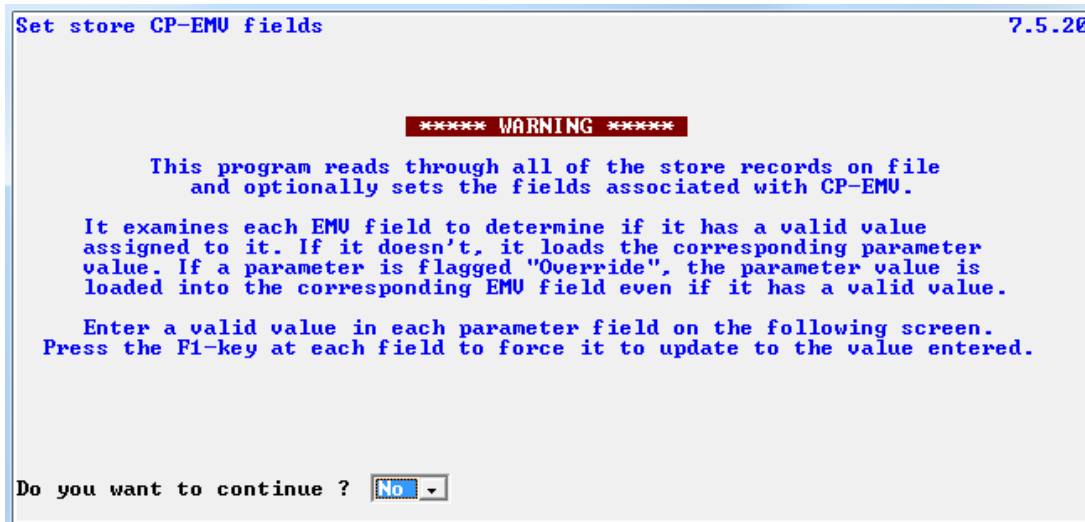
## File Utilities>Special>System>Set CP-EMV fields

Since all of the setup fields associated with CP-EMV are stored on each individual store record you need to ensure each store record has valid values. If you have a single, store that's no problem at all. But what if you have dozens or hundreds of stores in your CounterPoint installation ? It would be quite time-consuming to manually evaluate all those store's CP-EMV configurations. If you wanted to make a change to all of your stores' configurations (such as changing the Figaro IP address) you'd have to update each store record manually. Can you trust the store managers to do this correctly ?

This program allows verifying and setting CP-EMV fields for all of your stores programmatically. After the initial "splash" screen you are presented a parameter screen containing each CP-EMV setup field. At each field you enter a valid value that would be used by most (if not all) of your stores. If any of your stores have an invalid value for the field, and the program is being run in "Update" mode, the parameter's value will be loaded into the store record's field. This ensures that even if the parameter value isn't exactly correct for that store, at least the field for that store will have a sensible value with predictive results.

**Having sensible values in each store's CP-EMV setup fields is critical.** If a store has random values in these fields the results could be anything from CounterPoint crashing to fake authorizations being created. You could run for days obtaining bogus authorizations for sales – not collecting actual credit card information – only to discover no funds were transferred to your merchant account during that period. Run this program (at least in Report-only mode) to determine if any of your stores have trash in their CP-EMV setup fields and to ensure they have the right values for the store.

When the program first starts it displays a warning screen.



Answering “Y” takes you to the parameter screen (seen here with some sample values entered).

```
Set store CP-EMU fields 7.5.20

Please enter:

  1. Report or update ?           Report-only
  2. Credit card gateway         FreedomPay      <Override>
  3. FCC Version number         4.2x           <Override>
  4. FreedomPay store-id        0123456789     <Override>
  5. FreedomPay terminal-id     9876543210     <Override>
  6. FCC Server IP address      123.123.123.123 <Override>
  7. FCC Server port            1012           <Override>
  8. FCC Server timeout <seconds> 150            <Override>
  9. Store signatures locally ?  Y              <Override>
 10. Floor limit                50             <Override>
 11. Operating mode             Normal          <Override>

Field number to change ? 
```

Parameter “**1. Report or Update ?**” controls whether the program is going to update the store records it processes or merely report what would happen. You should run the program in report-only mode first to identify the condition of your current data and the changes that would be made.

Parameter “**2. Credit card gateway**” controls whether the store(s) would be processing their credit cards through CP-Gateway or FreedomPay. It might seem to be a pointless parameter since the whole point of this is to utilize EMV credit cards. But what if you have 100 stores in your CounterPoint installation and only a couple are going to start using CP-EMV at first ? You’ll want to set this parameter to “CP-Gateway” to get 98 stores set correctly then change the other 2 stores to FreedomPay manually.

Parameter “**3. FCC Version number**” sets the version of FreedomPay’s software the store(s) will be using. CP-EMV supports FCC versions 3.x, 4.0x, 4.1x and 4.2x.

Parameter “**4. FreedomPay store-id**” needs to have a value even if it’s the wrong one for the store. Each store will probably want to have its own unique store and terminal ids. But It’s better to have the wrong value here and be able to process cards (and figure out who owes who what later) than to not be able to process cards at all,

Parameter “**5. FreedomPay terminal-id**” is the second half of the merchant’s identification. Set it to a valid value that is related to the store-id field above. (i.e. Don’t try to mix-and-match store and terminal ids.)

Parameter “**6. FreedomPay server IP address**” is the network address the store(s) will use to send authorization requests to Figaro. If you’ve read this far in this document you already know what this is.

Parameter “**7. FreedomPay port**” is the port Figaro listens on for incoming authorization requests.

Parameter “**8. FreedomPay timeout (seconds)**” sets the number of seconds CounterPoint will wait for a response. There’s an entire section explaining the purpose and use of this parameter in the configuration section of this document. (As are all of these parameters.)



Parameter **“9. Store signatures locally”** is a Yes/No field to control whether CP-EMV will decode signature captures and store them locally (...\\SYDATA\\IMAGES).

Parameter **“10. Floor limit”** determines the dollar amount at which an authorization requires a signature.

Parameter **“11. Operating mode”** indicates whether the Ingenicos are attached to each register via a USB cable or if they are connected to the local network via an Ethernet cable.

Parameters 2 through 11 allow pressing <F1> to “Override” the parameter. Typically, the program will only update a store’s CP-EMV field if that field contains an invalid value. For example, let’s say a store record currently has the value “3.x” for the FCC Version number field and you enter “4.2x” for the parameter’s value without overriding the parameter. When the program runs it will not update the FCC Version number field on the store record because the store record does have a valid value (3.x) for that field. If there was some other value (any value other than one of the valid values) the program would load the parameter value “4.2x” into the store’s FCC Version number field.

But what if you really want to load 4.x into all of your store’s records ? Such as you just installed version 4.2x of our software and you want to turn the new version on for all of your stores. In this case you would press <F1> at the FCC Version number field to turn Override on. The parameter value would then be loaded into every store record on file. Overriding works this way for parameters 2 through 10.

Opposite of that is the option to ignore a field’s value. It kind of defeats the purpose of this program (to ensure all store CP-EMV fields have a valid value), but pressing <F2> at a field tells the program not to alter that field on any of the store records. That’s not to say the resulting report isn’t useful. All fields on the original store record are printed on the resulting report. If the field being ignored has an invalid value it will still be shown on the report. But if the field is being ignored it will not be fixed.

There are some fields that allow a wide range of valid values such as the FCC Server IP address and the store and terminal ids. There’s really no way to know if these fields have valid values. The program checks to see if the IP address field contains a properly formatted IP address and that numeric fields have numeric values in them.

As the program runs it creates a report. The first page of the report is the parameters with which it was run. For each store processed the program prints a single line showing the current values for the EMV fields as they are in the store record. If any field on the record would be changed a second line is printed containing just the fields that would be changed. Fields that would not be changed are not displayed on this second line. If the program was run in “Report-only” mode the literal “(different)” is printed in left margin. If the program is run in “Update” mode the literal “(changed)” is printed. See the end of this document for a [sample](#) of this report.

## CounterPoint Startup Program

When you start CounterPoint the program initially assumes the latest possible date is 2019. It's not until you log in that it even attempts to calculate the actual date. The problem is if you're launching CounterPoint to run an unattended program. In that case no one ever logs in and so that century-end date of 2019 gets passed to whatever program being run. If that program doesn't recalculate the date itself, any date fields in any records created during the run have the century-end date of 2019. So basically the dates all have 19 in their century portion of their dates.

The EMV transaction history file format changed between versions 3.x and 4.0x. We modified the start up program to check the format of the current EMV file and convert it to the new format.

On a more playful note we moved things around on the startup screen enough to proudly splash our name all over it. If you're running CounterPoint in a character-based environment you'll see:

```
-----
SYNCHRONICS CounterPoint XXXXXXXXXXXXXXXXXXXX          Company name goes here

Copyright (C) 1995-XXXX Radiant Systems, Inc. and its licensors.
All rights reserved.
EMV Credit card processing by Infinity Software Solutions, Inc.
+-----+
| This software registered for exclusive use by: |
|           This software is not registered      |
|           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    |
|           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    |
|           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    |
|           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    |
|           XXXXXXXXXXXXXXXXXXXX XX XXXXXXXXXX  |
|           Temporary registration expires on 99/99/99 |
|           This is a demonstration copy        |
+-----+
*** Modification libraries active ***

User ID   XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Password *****
Workgroup None XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Any change ? _
-----
```

Now we're famous !

## SCREEN Console I/O Program

When you define a new record in CounterPoint you add all of the fields you know you're going to need then add a chunk of extra space at the end of the record called FILLER. You create the FILLER at the end to add fields to later as the need arises. You have to do that because once the file has been created using the current record layout you cannot change the size or indices of the file without having to redefine the file again and creating a program to convert to file to the new format.

When you add a new field to an existing record layout you redefine part of the trailing FILLER to be the new field and you reduce the size of FILLER. Typically you add fields at the very end of the record and use the available space towards the start of the record. That's always been the best strategy because Synchronics usually adds fields right after the record's last existing field. So you end up with something like this.

```
...
      03 CUST-ECOMMERCE-PERM-ACTIV-FLG      PIC X(1) .
      03 CUST-USE-EMAIL-IN-MKTNG-FLG      PIC X(1) .
          88 USE-EMAIL-IN-MKTNG          VALUES SPACES, "Y" .
      03 CUST-MKTNG-DAT                    PIC 9(8) .
ISS969* 03 FILLER                          PIC X(5) .
ISS969  03 FILLER                          PIC X(4) .
ISS969  03 CUST-ALLOW-TOKENS-FLG         PIC X(1) .
ISS969      88 CUST-ALLOWS-TOKENS        VALUE "Y" .
```

This line has been commented out

In this example we commented out Synchronics' existing 5 byte FILLER (in gray) and replaced it with our own 4 byte FILLER and added a new field named CUST-ALLOW-TOKENS-FLG at the end of the record.

This works great, but there's a problem. In COBOL (the computer language CounterPoint is written in) FILLER fields are not addressable (so you cannot assign them a value) nor are they initialized when you use the INITIALIZE command to clear the record's contents. If you move SPACES to the entire record first then INITIALIZE the record they do get set to spaces. That works well for FILLER fields that occur in the middle of the record. (FILLERS can actually be placed anywhere within a record's definition.) But when the FILLER is at the end of the record layout it is stripped off when the record is written to disk. This saves disk space. But when the record is read back in later the value of the bytes in the trailing FILLER are back to being "undefined".

Normally this doesn't matter because FILLER fields are unaddressable and the program can't do anything with them anyway. The problem occurs when you add a new field in what was once FILLER area of a record. None of the existing records on disk have any values for the area where the new field resides. So when the runtime reads such a record from disk and copies the record's data byte-by-byte into its internal record buffer, the old record is shorter than the new record's definition and it leaves whatever random characters were in the buffer at that location of the new field. These characters could be normal text characters you can type – not ! That actually rarely happens. Generally that area of the record is going to have some unkeyable, undisplayable garbage that you'd never know was there. But it's there.

This can play havoc when that field is used in logic statements like “IF CUST-ALLOW-TOKENS-FLG = SPACES” (drawing on the example record above). Normally such a field will have a space, a “Y” or an “N” character in it. A programmer working with a new field must be very careful how they use the field. You cannot assume the field will have sensible values. You can’t even print such a field. It may a form feed character in it that ejects a page in the middle of a report. Or it might have a control code that shifts the printer into portrait mode. The value of the new field must be tested for “trash” before displaying or printing it.

Any program that writes such expanded records needs to be recompiled with the new record definition – including file utilities. This will force all newly written records to have proper values in the new field. Failing to recompile a program that writes new records will result in continuing to create records with trash in them. Programs that merely read the file and don’t need to access the new field don’t need to be recompiled.

Having explained all that let’s get to the problem at hand. You have probably noticed that when you enter a field in CounterPoint it may “echo” the existing contents of the field to the screen. That is very desirable for many fields – especially long fields or fields that carry strange values. It’s quite annoying to have to rekey a 50-character field merely because you accidentally entered a comma character instead of a period. That ability to echo a field is the “gotcha”. The CounterPoint program that accepts keyboard input and displays things to your screen is named SCREEN. It doesn’t discriminate between trash characters and valid text. When it echoes a field to the screen it actually echoes the trash characters to the screen as well. Unless the trash character means something to the monitor’s display driver (which is quite unlikely) the operator would see absolutely nothing on the screen to clue them in that there’s trash on the screen. Merely pressing the <ENTER> key will send that trashy field right back to SCREEN which will gladly accept the value.

Let’s say you’re entering an echoed field that is seemingly blank, but it in fact has trash in it. Let’s say you’re keying in the new store-id field that is part of your CP-EMV set up. After setting everything up you just can’t get credit cards to work. You get errors like “Invalid merchant credentials” or “Invalid request parameters”. After hours of checking and double-checking you reenter your store-id and everything starts working perfectly. The problem was trash in the new store-id field. I’ve seen this exact error several times. That’s why I advised you to fill the rest of the field with spaces all the way to the end when you’re entering the fields during the set up.

To end this forever we modified SCREEN to check the contents of all entry fields being echoed to the screen. Character by character, if any of the characters are not a character that can be keyed they are replaced with a space character. Thus it is now impossible for any field – even an echoed field – to be keyed with trash in it anywhere in CounterPoint.

Previously, we added the ability to play a user-defined sound file depending on the results of an authorization request. You can find the details of that in [“Sounds”](#) in the Features section of this document. The code that does that is also in the SCREEN program.

Years ago we discovered a bug in CounterPoint's date calculation routine. We corrected the bug and sent Synchronics a description of the problem, a custom program that demonstrated the flaw and the source code to fix it and for the test program. Their response was "Prove to us that it actually affects our software and we'll fix it". So we did that. They then wrote their own code – different from ours – that was twice as long. It's like they went out of their way not to use our code even though our solution was eloquent and compact. Whatever. Imagine our surprise when we discovered the same bug was still in coded into the console I/O program ! You'd think they'd handle the bad code wherever it was located. Nope. So we fixed this program ourselves.

Over the years we've been stymied with coloring text in CounterPoint screens. CounterPoint allows 8 background colors and 16 foreground colors for a theoretical total of 128 combinations. Two of those combinations; #7 - light grey on black and #51 - cyan on cyan are overwritten with "hardcoded" values. Number 7 displays grey on the current background color. This is essential for displaying "(n/a)" (for example) for fields whose entry is disabled for specific reasons. Number 51 displays light red on black which is the rarely used color scheme for error messages.

The problem is how we want to draw attention to the value of a field as an indication that the value is good or bad ? While CounterPoint has this broad palette of color combinations what we really want is to display the text (the foreground) in the desired color with the background in the current color. Programmatically, we can choose to display the text as light red on light grey, but what if the user has chosen green or black as their background color ? Now our nice, new functionality has created a garish hole on the screen. Hardly impressive.

To correct the situation we added new functionality to SCREEN to return the current palette. This allows us to retrieve the foreground and background colors for borders, literals, entry fields and highlight fields. We can then mask the desired foreground color on the current background color to get the results we've always wanted. It only took us 19 years to do it.

Right now there's only a single field in all of CounterPoint that's using this feature - the "Tax amount" field in [System>Utilities>Ad hoc authorizations](#). But it's a start and now that you know it's available you might want us to make use of it elsewhere.

## **POS Receipts**

EMV has its own requirements for the information printed on the receipt. There are several new EMV-specific fields that must be printed. The requirements are dry reading. To alleviate this pain from you we've provided baseline forms for you to use or to model from. These forms are named:

EMV40 "EMV Plain paper 40 columns"

EMV80 "EMV Plain paper 80 columns"

The forms are provided in export file format in the file PSFRMF.EXP. To import the forms copy the file to your EXPORT directory and run the POS file utilities. Select "3. Restore from an export file" then select file "6. Forms file". When prompted select "Add records to existing file". That last step is very important. Don't select "Create new file" or our sample forms will be all that's in the file. All your existing forms will be immediately deleted.

## **Offline Ticket Entry**

If you have an instance of Offline Ticket Entry (OTE) running it is writing its information to its own separate set of data files. That includes EMV information records and any signature capture files created during authorizations (if saving them locally is turned on). During the upload process (when the offline site is sending its data back to the online system) these files need to be transmitted for incorporation into the enterprise's data. The best way to understand the process and so determine the changes you need to make to support EMV processing is to read CounterPoint's electronic documentation. Having said that let's cover some of the changes here.

### **Creating Workstation Files**

First you need to load the current data files from the online system to your offline system. The batch file ZOFFDAT.BAT zips the files listed in the file OFFDAT.RSP into the file OFFDAT.ZIP. This is the "vanilla" file that is used to seed the typical data files in the offline installation. You might think to merely add the EMV information file (SYEMVF) to OFFDAT.RSP, but that is not recommended since a future upgrade (or reinstall) could replace OFFDAT.RSP and your changes would be lost. Instead create a file named OFFDATU.RSP with a single line; SYDATA\SYEMVF.DAT. If you want to copy the online system's signature file add the line SYDATA\IMAGES\\*. \* as well. When ZOFFDAT.BAT is executed it will look to see if OFFDATU.RSP exists and if it does it will zip the file(s) listed within it into the file OFFDATU.ZIP. ZOFFDAT.BAT already knows about OFFDATU.ZIP and will copy it to the offline system for you.

To save you a little time (and thinking) we created an OFFDATU.RSP for you. The installation should have already copied it to your top-level directory. You can also find a copy in the DATAFILS directory in the CP-EMV installation set. If you already had an OFFDATU.RSP when you ran the CP-EMV installation script look for your original version renamed OFFDATU.RSP.EMV.

The paragraph above describes the changes to make to get the online EMV files into the offline system's data files. But you might not even care about that. An updated EMV information file isn't actually required to perform typical processing like entering tickets. It's only used when viewing history or open/closed items. The signature capture files can be several kilobytes each and can result in a large OFFDATU.ZIP file. So unless your offline system actually needs the ability to see this information you might want to skip adding the EMV information file (or at least the signature files) to the process entirely.

## Backup the Offline Data

Officially the first step when uploading offline data is to backup the offline data. That saves your butt of something were to go wrong during the process. (And I know you backup regularly right ?) As each record is extracted from the offline system's files its "dirty flag" is cleared so it won't be extracted again. If something was to go wrong and you needed to rerun "Create upload file" again, those records whose dirty flags had been cleared would not be included in the upload file.

When you select "Backup offline data" from the CounterPoint menu a batch file named OFFBAK.BAT is executed. That batch file copies the offline files whose dirty flags are about to be cleared to a directory named OFFBACK. If the create upload program is going to process EMV information records you should add the EMV file to the backup process. But don't just add the EMV file to OFFBAK.BAT (for the same reasons we recommend you don't modify OFFDAT.RSP). When OFFBAK.BAT executes it looks for and executes the batch file OFFBAKU.BAT (if it exists). So create a batch file named OFFBAKU.BAT and add the code to backup the EMV information file (SYDATA\SYEMVF.DAT). You can use OFFBAK.BAT as a model for the new batch file. The most important line in the new batch file would look like:

```
COPY SYDATA\SYEMVF.DAT OFFBACK\SYEMVF.DAT
```

To save you a little time we created an OFFBAKU.BAT for you. The installation should have already copied it to your top-level directory. You can also find a copy in the DATAFILS directory in the CP-EMV installation set. If you already had an OFFBAKU. BAT when you ran the CP-EMV installation script Look for your original version renamed OFFBAKU. BAT.EMV.

Shockingly, when we tested this system we found the vanilla ZOFFDAT.BAT had two bugs. When it called OFFBAKU.BAT it didn't pass the company-id to it. Of course that won't do since OFFBAKU.BAT needs the company-id to know where to find the data files to back up. Also, it used a program named SCHDATE.EXE to add the system date and time to the log file. That's a 16-bit program so it won't run on most current 64-bit operating systems. We replaced the use of SCHDATE.EXE with Windows built-in %DATE% and %TIME% environment variables. So we corrected ZOFFBAK.BAT and packaged it in as well. Of course this version might be overwritten by an update so take care.

Note I'm not mentioning adding a line to OFFBAKU.BAT to backup the signature files. That's because signature files are copied in their entirety. There are no dirty flags to indicate which records (files in this case) to copy. Instead "Create upload file" program copies the signature files related to the EMV information records it finds to upload. So there's no need to backup signature files here.

## Create the Upload File on the Offline System

Loading the latest-and-greatest version of the EMV information file to the offline workstation might be optional, but the opposite is not true. You need to extract newly created EMV information records from the offline system and add them to the online system or they will never be made available to the online system or worse might be lost permanently. But you don't need to make any changes to any scripts or .RSP files to make this happen. (Except for signature capture files, but more on that in a bit.) We modified the program that builds the OTE upload file to automatically include EMV information records. But if you're running OTE you're already going to run this program anyway right ?

Signature capture files are a different matter though. They are individual files as opposed to records that can be written to a single "upload" file. So we can't merely add them to the upload file to be copied along with the other new and changed records. Each new signature file has to be associated with an EMV transaction record. So as the "Create upload file" program processes each EMV record, it looks to see if there is an associated signature file. If there is, it copies the file to a new directory named OTE . This directory will be located off the "root" of the value entered in the "Upload file path" parameter in the "Create upload file" program. If you leave that parameter blank and allow the program to default to "(Top level)" the OTE directory will be located off the company's data directory like this "/syn/DEMO1/OTE". Once "Create upload file" has finished, the OTE directory should have all of the new signature files since the last import program ran. (More on that.)

So at this point the register's upload file (PSREG###.DAT) including new EMV records has been built and its signature files copied to the new OTE directory. These files could be on a separate file system (like a network drive or a flash drive) or be located on the offline system. If you use a script or a manually-keyed command to move the upload file to the online system, you need to remember to copy the signature files as well. We modified the import program to know how to locate and handle the signature files based on the "Upload file path" parameter, but there is no process to actually move the signature files from the offline system to the online system. So if you have a system for moving the upload program around you'll need to tweak it a bit to copy the signature capture files around as well.

## Upload File List

If you run "Upload file list" you might notice that no EMV records are reported. That's because the program only displays ticket information. There are actually other types of records hidden in the file that CounterPoint doesn't show you. Similarly, we decided not to confuse the report format by cluttering it with EMV information. You just have to trust us that it's in there.

## Upload the Offline Data

We modified the program that imports the offline data files to include the EMV information file and the signature capture files for you. There's not much to say about the process. For each new EMV records imported the program looks for and copies the related signature file to the normal directory for signature files (*company-id*\SYDATA\IMAGES).



## Multi-Site

CounterPoint's Multi-Site feature allows multiple CounterPoint installations to share their information. Some of this information can be quite important such as Accounts Receivable and gift certificate usage. Also shared are select fields in records like the customer file and whole records such as open items and ticket history. This allows all of the stores in an enterprise to "see" the same data regardless of the store that created it. As mentioned earlier (and repeatedly) we've added a new file (SYEMVF) to CounterPoint to retain the information returned with valid authorizations. There are several inquiry screens that allow displaying this information in connection with the area of CounterPoint to which it is associated. So one can press a function key in View Ticket History to see a tender's EMV information. Ditto for an A/R open item or an O/E deposit.

To allow all of the stores to access this EMV information we've added the SYEMVF file to CounterPoint's Multi-Site suite. This required modifying or recompiling a surprising number of programs (24 to be exact) to do it right. The reason is because Multi-Site allows quite a granular control over which packages are shared between the hub and the stores. We could have chosen to simply copy the EMV information for all types of transactions between all of the stores. That would have greatly simplified the process. But then all of the stores would have all of the EMV records from all stores; including stores that chose not to share their data for certain packages. So we chose to add EMV support that follow the way Multi-Site is set up. So if Multi-Site is configured to not share history, EMV information will not be transmitted to the hub.

There is no additional set up required to process EMV data. However, if you have scripts that copy individual data files around (other than the usual satellite files) you might need to add the SYEMVF file to them.

There aren't many screens displayed by Multi-Site as it runs. Each satellite has a progress screen as it builds its transmission file. This screen has counts for each type of record extracted. This screen resembles:

Search for changes 7.5.20

Currently searching: System  
Searching file: SYNSFF  
Changes detected: 0

Total changes detected: 752

POINT OF SALE	INVENTORY	ORDER ENTRY	SYSTEM
Orders 11	Items 23	Orders 0	Notes 2
Other 56	Inventory 38	Setup 0	Timecards 0
Setup 6	Barcodes 0	EMU 0	Bad checks 0
EMU 62	Kits 0		Setup 8
	Layers 0	OPEN TO BUY	
	Prices 1	Setup 0	
	Ser/lot 9		
	Vendors 3	LABELS	OTHER FILES
	Vend items 8	Setup 1	Control 4
	Markdowns 0		Users 2
	Trkd kits 0	PURCHASING	Mail 0
	Setup 6	P.O.'s 0	History 238
		Setup 0	Hist EMU 0

Search complete - Press ESC

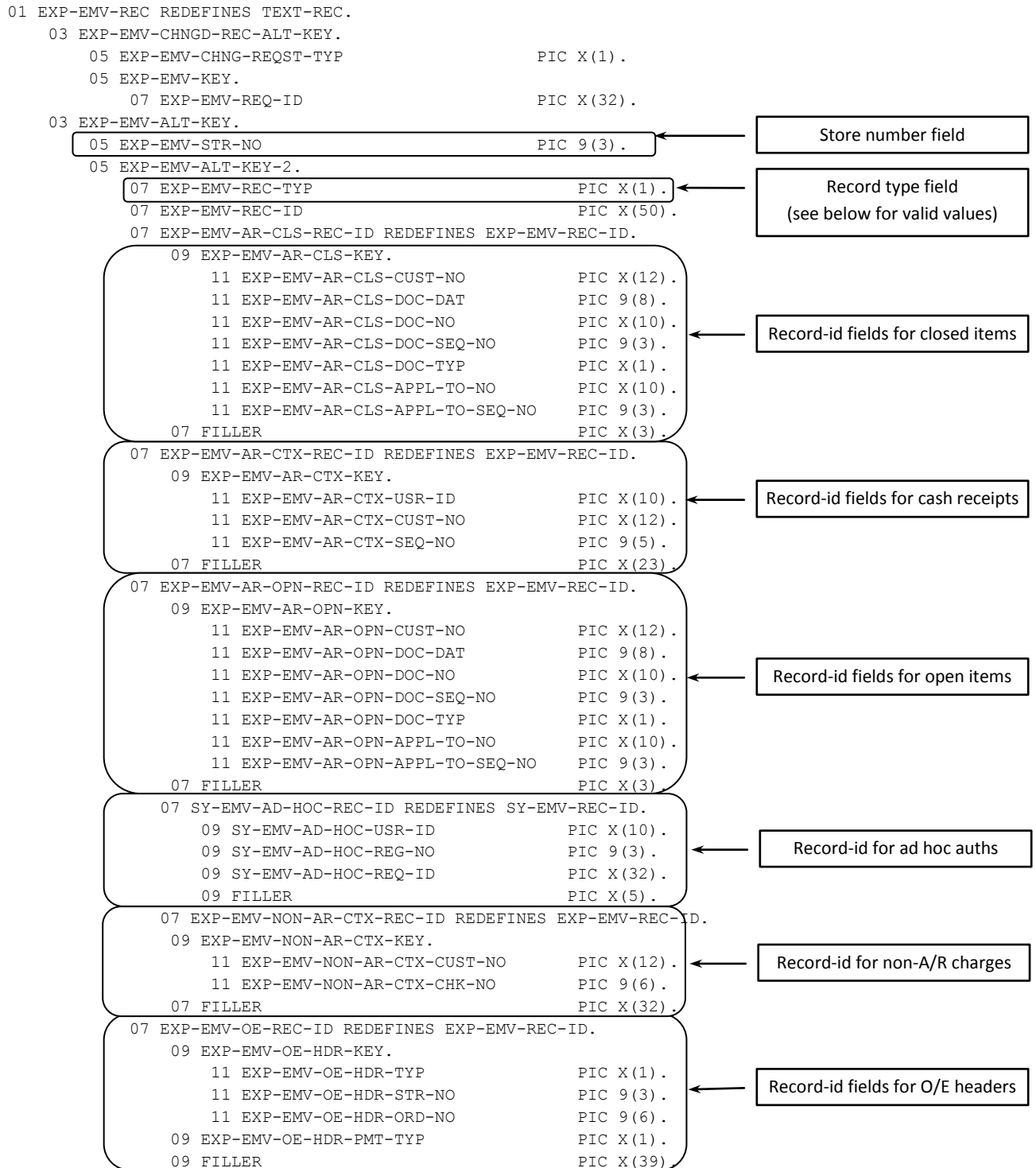
In the screenshot above you can see that we've added four new EMV counters (circled in red) to allow visually confirming the EMV records are being processed.

The program that imports the satellite files at the hub also displays the counts of the records being processed. However this screen didn't have room for us to add a separate counter for each record type. Besides, the EMV file is a system-wide file. So we merely added a single counter labeled "EMV" in the "System" quadrant of the screen. (Sorry – no screenshot for that yet.)

# Let's Get Geeky !

## The New EMV Information File

As mentioned earlier we created a new file in which to carry the additional EMV transaction data. The file is named SYEMVF and resides in the SYDATA directory. Below is the exported file format.



<pre> 07 EXP-EMV-PS-REC-ID REDEFINES EXP-EMV-REC-ID. 09 EXP-EMV-PS-HDR-KEY.     11 EXP-EMV-PS-HDR-TYP          PIC X(1) .     11 EXP-EMV-PS-HDR-REG-NO      PIC 9(3) .     11 EXP-EMV-PS-HDR-TICKET-NO  PIC 9(6) . 09 EXP-EMV-PS-PMT-IDX          PIC 9(2) . 09 FILLER                        PIC X(38) . </pre>	Record-id fields for POS headers
<pre> 07 EXP-EMV-PS-LIN-REC-ID REDEFINES EXP-EMV-REC-ID. 09 EXP-EMV-PS-LIN-KEY.     11 EXP-EMV-PS-LIN-HDR-KEY.         13 EXP-EMV-PS-LIN-TYP      PIC X(1) .         13 EXP-EMV-PS-LIN-REG-NO  PIC 9(3) .         13 EXP-EMV-PS-LIN-TICKET-NO PIC 9(6) .     11 EXP-EMV-PS-LIN-REC-TYP     PIC X(1) .     11 EXP-EMV-PS-LIN-LIN-NO      PIC 9(5) .     11 EXP-EMV-PS-LIN-SEQ-NO     PIC 9(5) . 09 FILLER                        PIC X(29) . </pre>	Record-id fields for POS lines (New for version 4.1)
<pre> 07 EXP-EMV-SA-HDR-REC-ID REDEFINES EXP-EMV-REC-ID. 09 EXP-EMV-SA-HDR-KEY.     11 EXP-EMV-SA-HDR-STR-NO      PIC 9(3) .     11 EXP-EMV-SA-HDR-TICKET-NO  PIC 9(6) .     11 EXP-EMV-SA-HDR-SEQ-NO     PIC 9(3) . 09 EXP-EMV-SA-PMT-IDX          PIC 9(2) . 09 FILLER                        PIC X(36) . </pre>	Record-id fields for history header
<pre> 07 EXP-EMV-SA-LIN-REC-ID REDEFINES EXP-EMV-REC-ID. 09 EXP-EMV-SA-LIN-KEY.     11 EXP-EMV-SA-LIN-HDR-KEY.         13 EXP-EMV-SA-LIN-STR-NO  PIC 9(3) .         13 EXP-EMV-SA-LIN-TICKET-NO PIC 9(6) .         13 EXP-EMV-SA-LIN-SEQ-NO  PIC 9(3) .     11 EXP-EMV-SA-LIN-LIN-REC-TYP PIC X(1) .     11 EXP-EMV-SA-LIN-LIN-NO      PIC 9(5) .     11 EXP-EMV-SA-LIN-LIN-SEQ-NO  PIC 9(5) . 09 FILLER                        PIC X(27) . </pre>	Record-id fields for history lines (New for version 4.1)
<pre> 07 EXP-EMV-TOKEN-REC-ID REDEFINES EXP-EMV-REC-ID. 09 EXP-EMV-TOKEN-CARD-CUST-NO   PIC X(12) . 09 EXP-EMV-TOKEN-CARD-SEQ-NO   PIC 9(2) . 09 EXP-EMV-TOKEN-CARD-NO       PIC X(20) . 09 FILLER                       PIC X(16) . </pre>	Record-id fields for token trxs (Changed for version 4.1)
<pre> 05 EXP-EMV-DAT          PIC 9(8) . 05 EXP-EMV-TIME         PIC 9(8) . </pre>	Trx date/time for all rec types

03 EXP-EMV-DATA.		
05 EXP-EMV-TRX-TYP	PIC X(1) .	**See below for valid values
05 EXP-EMV-MASKED-CC-NO	PIC X(20) .	
05 EXP-EMV-CARD-EXP-DAT	PIC X(4) .	
05 EXP-EMV-CARD-TYP	PIC X(10) .	
05 EXP-EMV-AUTH-COD	PIC X(8) .	
05 EXP-EMV-VOID-REFER-REQ-ID	PIC X(32) .	
05 EXP-EMV-REQ-AMT	PIC S9(9)V99 SIGN IS TRAILING SEPARATE.	
05 EXP-EMV-APPROVED-AMT	PIC S9(9)V99 SIGN IS TRAILING SEPARATE.	
05 EXP-EMV-PARTIAL-AMT	PIC S9(9)V99 SIGN IS TRAILING SEPARATE.	
05 EXP-EMV-TIP-AMT	PIC S9(9)V99 SIGN IS TRAILING SEPARATE.	
05 EXP-EMV-ACCT-BAL	PIC S9(9)V99 SIGN IS TRAILING SEPARATE.	
05 EXP-EMV-DECISION	PIC X(1) .	
05 EXP-EMV-SIG-AVAIL-IND	PIC X(1) .	
05 EXP-EMV-ISSUER-NAM	PIC X(20) .	
05 EXP-EMV-NAM-ON-CARD	PIC X(30) .	
05 EXP-EMV-ENTRY-MODE	PIC X(10) .	
05 EXP-EMV-LANE-ID	PIC 9(10) .	
05 EXP-EMV-ERR-COD	PIC 9(6) .	
05 EXP-EMV-FLOOR-LIMIT	PIC 9(9) .	
05 EXP-EMV-TRX-ID	PIC X(20) .	
05 EXP-EMV-MERCH-REF-COD	PIC X(20) .	
05 EXP-EMV-TOKEN	PIC X(28) .	

```

05 EXP-EMV-TOKEN-EXP-DAT          PIC 9(6) .
05 EXP-EMV-DCC-ACCEPTED-FLG       PIC X(1) .
05 EXP-EMV-EMV-CVM                PIC X(5) .
05 EXP-EMV-EMV-TAG-DATA.
   07 EXP-EMV-EMV-TAG-50           PIC X(20) .
   07 EXP-EMV-EMV-TAG-5F2A        PIC X(5) .
   07 EXP-EMV-EMV-TAG-5F34        PIC X(5) .
   07 EXP-EMV-EMV-TAG-82          PIC X(5) .
   07 EXP-EMV-EMV-TAG-95          PIC X(10) .
   07 EXP-EMV-EMV-TAG-9A          PIC X(10) .
   07 EXP-EMV-EMV-TAG-9C          PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F02        PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F03        PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F07        PIC X(5) .
   07 EXP-EMV-EMV-TAG-9F0D        PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F0E        PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F0F        PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F10        PIC X(15) .
   07 EXP-EMV-EMV-TAG-9F12        PIC X(20) .
   07 EXP-EMV-EMV-TAG-9F1A        PIC X(5) .
   07 EXP-EMV-EMV-TAG-9F26        PIC X(20) .
   07 EXP-EMV-EMV-TAG-9F27        PIC X(5) .
   07 EXP-EMV-EMV-TAG-9F34        PIC X(10) .
   07 EXP-EMV-EMV-TAG-9F36        PIC X(5) .
   07 EXP-EMV-EMV-TAG-9F37        PIC X(10) .
   07 EXP-EMV-EMV-TAG-DF03        PIC X(10) .
   07 EXP-EMV-EMV-TAG-DF04        PIC X(10) .
   07 EXP-EMV-EMV-TAG-DF05        PIC X(10) .
   07 EXP-EMV-EMV-TAG-AID         PIC X(15) .
   07 EXP-EMV-EMV-TAG-ARC         PIC X(5) .
   07 EXP-EMV-EMV-TAG-IAD         PIC X(15) .
   07 EXP-EMV-EMV-TAG-TSI         PIC X(5) .
   07 EXP-EMV-EMV-TAG-TVR         PIC X(10) .
   07 EXP-EMV-EMV-TAG-TAC-DEFAULT PIC X(10) .
   07 EXP-EMV-EMV-TAG-TAC-DENIAL  PIC X(10) .
   07 EXP-EMV-EMV-TAG-TAC-ONLINE  PIC X(10) .

05 EXP-EMV-OFFLINE-RESP-FLG       PIC X(1) .
05 EXP-EMV-EMV-CVM-METH            PIC X(15) .
05 EXP-EMV-REQ-GUID                PIC X(36) .
05 EXP-EMV-PREV-REQ-ID             PIC X(32) .
05 EXP-EMV-PIN-VERIFIED-FLG        PIC X(1) .
05 EXP-EMV-SIG-REQD-FLG            PIC X(1) .
05 SY-EMV-CUST-NO                  PIC X(12) .
05 SY-EMV-TRX-DELETED-FLG          PIC X(1) .

```

\* A=Cash receipt, B=A/R Open item, C=A/R Closed item, H=Ad hoc trx, N=non-A/R cash receipt, O=O/E Header, P=POS Header, L=POS Line item, S=Ticket history header, U=Ticket history line item, T=Token

\*\* S=Sale, R=Return, V=Void, T=Create token, A=SVC Activate, I=SVC Issue, B=SVC balance Inquiry, C=SVC Recharge, K=SVC Cash back, P=SVC Tender, F=SVC Void, M=SVC Merchant return

If you study the file description above you'll see the REC-ID field has been redefined 11 different ways. One for each type of record the file retains EMV information for. The fields' names should be sufficiently descriptive to convey each of their purposes. If you intend to use this file we suggest you export the data and/or view the records via Pervasive Control Center (or another ODBC-capable program) to familiarize yourself with the data.

## Changes to Existing CounterPoint Files

In addition to adding a new file to Counterpoint we've added many new fields to several existing CounterPoint files. As a rule, we always add new fields at the end of the record layout with subsequent fields added towards the beginning of the record layout. We're not going to copy all of the full file descriptions for all of the record layouts we've changed. We can make those available to you upon request. Unless you're writing programs that can directly access the CounterPoint data files you probably don't really need to know anyway. You can use the section below to identify the files (tables) we modified and the fields we added.

## CounterPoint SQL Connection

### Changes to Existing Tables

In order to make this enhancement work we had to add quite a few fields to several of CounterPoint's file layouts. We added these new fields to the Windows (for now) CounterPoint's "Complete" SQL Connection dictionary. We also added the new EMV file. Adding these fields probably won't help you much in your day-to-day use of CounterPoint, but if you have created any Crystal reports or extraction routines that access payment information they might prove quite useful. Here's an overview of the files we changed and the new fields we added.

<u>Table</u>	<u>Field-name</u>	<u>Type</u>	<u>Size</u>
AR_CLS	EMV_PMT_FLG	AlphaNumeric	1
AR_CLS	EMV_REQ_ID	AlphaNumeric	32
AR_CTL	DFLT_CC_REG_NO	Numeric	3 integers
AR_CTL	SAV_TOKENS_FLG	Alphanumeric	1
AR_CTX	EMV_ACCT_BAL	Numeric	7 integers, 2 decimals
AR_CTX	EMV_PMT_FLG	AlphaNumeric	1
AR_CTX	EMV_REQ_ID	AlphaNumeric	32
AR_CTX	EMV_TOKEN	AlphaNumeric	20
AR_OPN	EMV_PMT_FLG	AlphaNumeric	1
AR_OPN	EMV_REQ_ID	AlphaNumeric	32
CUST	ALLOW_TOKENS_FLG	AlphaNumeric	1
OE_CTL	DFLT_CC_REG_NO	Numeric	3 integers
OE_HDR	BAL_EMV_PMT_FLG	AlphaNumeric	1
OE_HDR	BAL_EMV_REQ_ID	AlphaNumeric	32
OE_HDR	BAL_EMV_TOKEN	AlphaNumeric	20
OE_HDR	DEP_EMV_PMT_FLG	AlphaNumeric	1
OE_HDR	DEP_EMV_REQ_ID	AlphaNumeric	32
OE_HDR	DEP_EMV_TOKEN	AlphaNumeric	20
PS_CARD	TOKEN	Numeric	7 integers, 2 decimals
PS_CARD	TOKEN_EXP_DAT	Numeric	8 integers
PS_HDR	EMV_ACCT_BAL_1	Numeric	7 integers, 2 decimals
PS_HDR	EMV_PMT_FLG_1	AlphaNumeric	1
PS_HDR	EMV_ACCT_BAL_2	Numeric	7 integers, 2 decimals
PS_HDR	EMV_PMT_FLG_2	AlphaNumeric	1
PS_HDR	EMV_ACCT_BAL_3	Numeric	7 integers, 2 decimals
PS_HDR	EMV_PMT_FLG_3	AlphaNumeric	1
PS_HDR	EMV_ACCT_BAL_4	Numeric	7 integers, 2 decimals
PS_HDR	EMV_PMT_FLG_4	AlphaNumeric	1
PS_HDR	EMV_ACCT_BAL_5	Numeric	7 integers, 2 decimals
PS_HDR	EMV_PMT_FLG_5	AlphaNumeric	1
PS_HDR	EMV_ACCT_BAL_6	Numeric	7 integers, 2 decimals

PS_HDR	EMV_PMT_FLG_6	AlphaNumeric	1
PS_LIN	SVC_REQ_ID	AlphaNumeric	32
PS_STR	EMV_FCC_IP_ADDR	AlphaNumeric	20
PS_STR	EMV_FCC_PORT	Numeric	5 integers
PS_STR	EMV_FCC_TIMEOUT	Numeric	3 integers
PS_STR	EMV_FCC_VER_NO	AlphaNumeric	1
PS_STR	EMV_FLOOR_LIMIT	Numeric	7 integers
PS_STR	EMV_OP_MOD_FLG	AlphaNumeric	1
PS_STR	EMV_SAV_SIGCAP_FLG	AlphaNumeric	1
PS_STR	EMV_STR_ID	AlphaNumeric	16
PS_STR	EMV_TERM_ID	AlphaNumeric	16
PS_STR	GATEWAY_FLG	AlphaNumeric	1
SA_HDR	EMV_PMT_FLG_1	AlphaNumeric	1
SA_HDR	EMV_PMT_FLG_2	AlphaNumeric	1
SA_HDR	EMV_PMT_FLG_3	AlphaNumeric	1
SA_HDR	EMV_PMT_FLG_4	AlphaNumeric	1
SA_HDR	EMV_PMT_FLG_5	AlphaNumeric	1
SA_HDR	EMV_PMT_FLG_6	AlphaNumeric	1
SA_LIN	SVC_REQ_ID	AlphaNumeric	32

### The New EMV Information File

Below is a separate description for the dictionary's view of the new SY\_EMV information table.

Field-name	Aliases	Type	Size	
Z_KEY_0_COMP_0	KEY_FLD REQ_ID	AlphaNumeric	32	← Primary key
Z_KEY_1_COMP_0	CHNG_REQST_TYP	AlphaNumeric	1	← CHNGD_REC_ALT_KEY
Z_KEY_1_COMP_1	Z_KEY_0_COMP_0	AlphaNumeric	32	
Z_KEY_2_COMP_0	STR_NO	Numeric	3	* See below for valid values
Z_KEY_2_COMP_1	TYP	AlphaNumeric	1	
Z_KEY_2_COMP_2	ID	AlphaNumeric	50	
	AD_HOC_REC_ID			
	AR_CLS_REC_ID			
	AR_OPN_REC_ID			
	AR_CTX_REC_ID			
	CUST_REC_ID			
	OE_REC_ID			
	PS_REC_ID			
	SA_REC_ID			
	TOKEN_REC_ID			
	Z_KEY_1_COMP_2			
Z_KEY_2_COMP_3	DAT	Numeric	8	← ALT_KEY
Z_KEY_2_COMP_4	TIME_FLD	Numeric	8	
Z_KEY_3_COMP_0	TYP	AlphaNumeric	1	
Z_KEY_3_COMP_1	Z_KEY_2_COMP_2	Alphanumeric	50	
Z_KEY_3_COMP_2	Z_KEY_2_COMP_3	Numeric	8	
Z_KEY_3_COMP_3	Z_KEY_2_COMP_4	Numeric	8	
Z_KEY_4_COMP_0	Z_KEY_2_COMP_0	Numeric	3	
Z_KEY_4_COMP_1	Z_KEY_2_COMP_3	Numeric	8	
Z_KEY_4_COMP_2	Z_KEY_2_COMP_4	Numeric	8	
Z_KEY_5_COMP_0	Z_KEY_2_COMP_3	Numeric	8	
Z_KEY_5_COMP_1	Z_KEY_2_COMP_4	Numeric	8	
				← ALT_KEY_2
				← ALT_KEY_3
				← ALT_KEY_4

<u>Field-name</u>	<u>Type</u>	<u>Length</u>	<u>Member of</u>
<b>AD_HOC_KEY</b>	AlphaNumeric	45	
AD_HOC_USR_ID	AlphaNumeric	10	AD_HOC_KEY, AD_HOC_REC_ID
AD_HOC_REG_NO	Numeric	3	AD_HOC_KEY, AD_HOC_REC_ID
AD_HOC_REQ_ID	AlphaNumeric	32	AD_HOC_KEY, AD_HOC_REC_ID
<b>AR_CLS_KEY</b>	AlphaNumeric	47	
AR_CLS_CUST_NO	AlphaNumeric	12	AR_CLS_KEY, AR_CLS_REC_ID
AR_CLS_DOC_DAT	Numeric	8	AR_CLS_KEY, AR_CLS_REC_ID
AR_CLS_DOC_NO	AlphaNumeric	10	AR_CLS_KEY, AR_CLS_REC_ID
AR_CLS_DOC_SEQ_NO	Numeric	3	AR_CLS_KEY, AR_CLS_REC_ID
AR_CLS_DOC_TYP	AlphaNumeric	1	AR_CLS_KEY, AR_CLS_REC_ID
AR_CLS_APPL_TO_NO	AlphaNumeric	10	AR_CLS_KEY, AR_CLS_REC_ID
AR_CLS_APPL_TO_SEQ_NO	Numeric	3	AR_CLS_KEY, AR_CLS_REC_ID
<b>AR_CTX_KEY</b>	AlphaNumeric	27	
AR_CTX_USR_ID	AlphaNumeric	10	AR_CTX_KEY, AR_CTX_REC_ID
AR_CTX_CUST_NO	AlphaNumeric	12	AR_CTX_KEY, AR_CTX_REC_ID
AR_CTX_SEQ_NO	Numeric	5	AR_CTX_KEY, AR_CTX_REC_ID
<b>AR_OPN_KEY</b>	AlphaNumeric	47	
AR_OPN_CUST_NO	AlphaNumeric	12	AR_OPN_KEY, AR_OPN_REC_ID
AR_OPN_DOC_DAT	Numeric	8	AR_OPN_KEY, AR_OPN_REC_ID
AR_OPN_DOC_NO	AlphaNumeric	10	AR_OPN_KEY, AR_OPN_REC_ID
AR_OPN_DOC_SEQ_NO	AlphaNumeric	10	AR_OPN_KEY, AR_OPN_REC_ID
AR_OPN_DOC_TYP	AlphaNumeric	1	AR_OPN_KEY, AR_OPN_REC_ID
AR_OPN_APPL_TO_NO	AlphaNumeric	10	AR_OPN_KEY, AR_OPN_REC_ID
AR_OPN_APPL_TO_NO_SEQ_NO	Numeric	3	AR_OPN_KEY, AR_OPN_REC_ID
<b>NON_AR_CTX_KEY</b>	AlphaNumeric	18	
NON_AR_CTX_CUST_NO	AlphaNumeric	12	NON_AR_CTX_KEY, NON_AR_CTX_REC_ID
NON-AR-CTX-CHK-NO	Numeric	6	NON_AR_CTX_KEY, NON_AR_CTX_REC_ID
<b>OE_HDR_KEY</b>	AlphaNumeric	10	
OE_HDR_TYP	AlphaNumeric	1	OE_HDR_KEY, OE_HDR_REC_ID
OE_HDR_STR_NO	Numeric	3	OE_HDR_KEY, OE_HDR_REC_ID
OE_HDR_ORD_NO	Numeric	6	OE_HDR_KEY, OE_HDR_REC_ID
OE_HDR_PMT_TYP	AlphaNumeric	1	OE_HDR_REC_ID
<b>PS_HDR_KEY</b>	AlphaNumeric	11	
PS_HDR_TYP	AlphaNumeric	1	PS_HDR_KEY, PS_HDR_REC_ID
PS_HDR_REG_NO	Numeric	3	PS_HDR_KEY, PS_HDR_REC_ID
PS_HDR_TICKET_NO	Numeric	6	PS_HDR_KEY, PS_HDR_REC_ID
PS_PMT_IDX	Numeric	1	PS_HDR_REC_ID
<b>PS_LIN_KEY</b>	AlphaNumeric	21	
PS_LIN_TYP	AlphaNumeric	1	PS_LIN_KEY, PS_LIN_REC_ID, PS_LIN_HDR_KEY
PS_LIN_REG_NO	Numeric	3	PS_LIN_KEY, PS_LIN_REC_ID, PS_LIN_HDR_KEY
PS_LIN_TICKET_NO	Numeric	6	PS_LIN_KEY, PS_LIN_REC_ID, PS_LIN_HDR_KEY
PS_LIN_LIN_TYP	AlphaNumeric	1	PS_LIN_KEY, PS_LIN_REC_ID
PS_LIN_LIN_NO	Numeric	5	PS_LIN_KEY, PS_LIN_REC_ID
PS_LIN_SEQ_NO	Numeric	5	PS_LIN_KEY, PS_LIN_REC_ID



<b>SA_HDR_KEY</b>	AlphaNumeric	14	
SA_HDR_STR_NO	Numeric	3	SA_HDR_KEY, SA_HDR_REC_ID
SA_HDR_TICKET_NO	Numeric	6	SA_HDR_KEY, SA_HDR_REC_ID
SA_HDR_SEQ_NO	Numeric	3	SA_HDR_KEY, SA_HDR_REC_ID
SA_PMT_IDX	Numeric	1	SA_HDR_REC_ID
<b>SA_LIN_KEY</b>	AlphaNumeric	14	
SA_LIN_STR_NO	Numeric	3	SA_HDR_KEY, SA_HDR_REC_ID, SA_LIN_HDR_KEY
SA_LIN_TICKET_NO	Numeric	6	SA_HDR_KEY, SA_HDR_REC_ID, SA_LIN_HDR_KEY
SA_LIN_SEQ_NO	Numeric	3	SA_HDR_KEY, SA_HDR_REC_ID, SA_LIN_HDR_KEY
SA_LIN_LIN_REC_TYP	AlphaNumeric	1	SA_LIN_KEY, SA_LIN_REC_ID
SA_LIN_LIN_NO	Numeric	5	SA_LIN_KEY, SA_LIN_REC_ID
SA_LIN_LIN_SEQ_NO	Numeric	5	SA_LIN_KEY, SA_LIN_REC_ID
<b>TOKEN_REC_KEY</b>	AlphaNumeric	24	
TOKEN_CARD_CUST_NO	AlphaNumeric	12	TOKEN_REC_ID
TOKEN_CARD_SEQ_NO	Numeric	2	TOKEN_REC_ID
TOKEN_CARD_CARD_NO	AlphaNumeric	20	TOKEN_REC_ID
TRX_TYP	AlphaNumeric	1	**See below for valid values
MASKED_CC_NO	AlphaNumeric	20	
CARD_EXP_DAT	AlphaNumeric	4	
CARD_TYP	AlphaNumeric	10	
AUTH_COD	AlphaNumeric	8	
VOID_REFER_REQ_ID	AlphaNumeric	32	
REQ_AMT	Numeric	9 integers, 2 decimals	
APPROVED_AMT	Numeric	9 integers, 2 decimals	
PARTIAL_AMT	Numeric	9 integers, 2 decimals	
TIP_AMT	Numeric	9 integers, 2 decimals	
ACCT_BAL	Numeric	9 integers, 2 decimals	
DECISION	AlphaNumeric	1	
SIG_AVAIL_IND	AlphaNumeric	1	
ISSUER_NAM	AlphaNumeric	20	
NAM_ON_CARD	AlphaNumeric	30	
ENTRY_MODE	AlphaNumeric	10	
LANE_ID	Numeric	10 integers	
ERR_COD	Numeric	6 integers	
FLOOR_LIMIT	Numeric	9 integers	
TRX_ID	AlphaNumeric	20	
MERCH_REF_COD	AlphaNumeric	20	
TOKEN	AlphaNumeric	28	
TOKEN_EXP_DAT	Numeric	8	
DCC_ACCEPTED_FLG	AlphaNumeric	1	
EMV_CVM	AlphaNumeric	5	
EMV_TAG_50	AlphaNumeric	20	
EMV_TAG_5F2A	AlphaNumeric	5	
EMV_TAG_5F34	AlphaNumeric	5	
EMV_TAG_82	AlphaNumeric	5	
EMV_TAG_95	AlphaNumeric	10	
EMV_TAG_9A	AlphaNumeric	10	
EMV_TAG_9C	AlphaNumeric	10	
EMV_TAG_9F02	AlphaNumeric	10	
EMV_TAG_9F03	AlphaNumeric	10	
EMV_TAG_9F07	AlphaNumeric	5	
EMV_TAG_9F0D	AlphaNumeric	10	
EMV_TAG_9F0E	AlphaNumeric	10	
EMV_TAG_9F0F	AlphaNumeric	10	

EMV_TAG_9F10	AlphaNumeric	15
EMV_TAG_9F12	AlphaNumeric	20
EMV_TAG_9F1A	AlphaNumeric	5
EMV_TAG_9F26	AlphaNumeric	20
EMV_TAG_9F27	AlphaNumeric	5
EMV_TAG_9F34	AlphaNumeric	10
EMV_TAG_9F36	AlphaNumeric	5
EMV_TAG_9F37	AlphaNumeric	10
EMV_TAG_DF03	AlphaNumeric	10
EMV_TAG_DF04	AlphaNumeric	10
EMV_TAG_DF05	AlphaNumeric	10
EMV_TAG_AID	AlphaNumeric	15
EMV_TAG_ARC	AlphaNumeric	5
EMV_TAG_IAD	AlphaNumeric	15
EMV_TAG_TSI	AlphaNumeric	5
EMV_TAG_TVR	AlphaNumeric	10
EMV_TAG_TAC_DEFAULT	AlphaNumeric	10
EMV_TAG_TAC_DENIAL	AlphaNumeric	10
EMV_TAG_TAC_ONLINE	AlphaNumeric	10
OFFLINE_RESP_FLG	AlphaNumeric	1
EMV_CVM_METH	AlphaNumeric	15
STR_NO	Numeric	3 integers
EMV-REQ-GUID	AlphaNumeric	36
EMV-PREV-REQ-ID	AlphaNumeric	32
EMV-PIN-VERIFIED-FLG	AlphaNumeric	1
EMV-SIG-REQD-FLG	AlphaNumeric	1
EMV-CUST_NO	AlphaNumeric	12
EMV-TRX-DELETED-FLG	AlphaNumeric	1

\* A=Cash receipt, B=A/R Open item, C=A/R Closed item, H=Ad hoc trx, N=non-A/R cash receipt, O=O/E Header, P=POS Header, L=POS Line item, S=Ticket history header, U=Ticket history line item, T=Token

\*\* S=Sale, R=Return, V=Void, T=Create token, A=SVC Activate, I=SVC Issue, B=SVC balance Inquiry, C=SVC Recharge, K=SVC Cash back, P=SVC Tender, F=SVC Void, M=SVC Merchant return

## Debugging

There are a few places to look for information and test when things aren't working right.

- The message on CounterPoint's screen
- The message on the Ingenico's screen
- Figaro's log file
- Susanna's log file
- The Windows Event Viewer for Figaro
- The Windows Event Viewer for Susanna
- Figaro's configuration file
- Susanna's configuration file
- The servers.xml configuration file
- CP-EMV's parameters in CounterPoint's store configuration
- CounterPoint's CP-EMV.REG file
- The PAL Ingenico update files

When Susanna is started with a proper configuration and the Ingenico is connected, configured and powered on you should see "Secured by FreedomPay" on the Ingenico's screen. If the Ingenico isn't connected to Susanna properly or either configuration is wrong the screen will display "This lane closed". But don't assume that if the Ingenico's screen is displaying "Secured by FreedomPay" that everything is copasetic between it and Susanna. If "Secured by FreedomPay" was already being displayed and the connection drops or Susanna is stopped the Ingenico's screen won't necessarily fall back to "This lane closed". You can even unplug the Ingenico's USB cable and the screen won't change. For example, you can establish a connection between Susanna and the Ingenico and the screen goes to "Secured by FreedomPay". Then stop Susanna and the Ingenico will continue to display "Secured by FreedomPay". Reboot the Ingenico and it will display "This lane closed". Start Susanna and the screen will change to "Secured by FreedomPay".

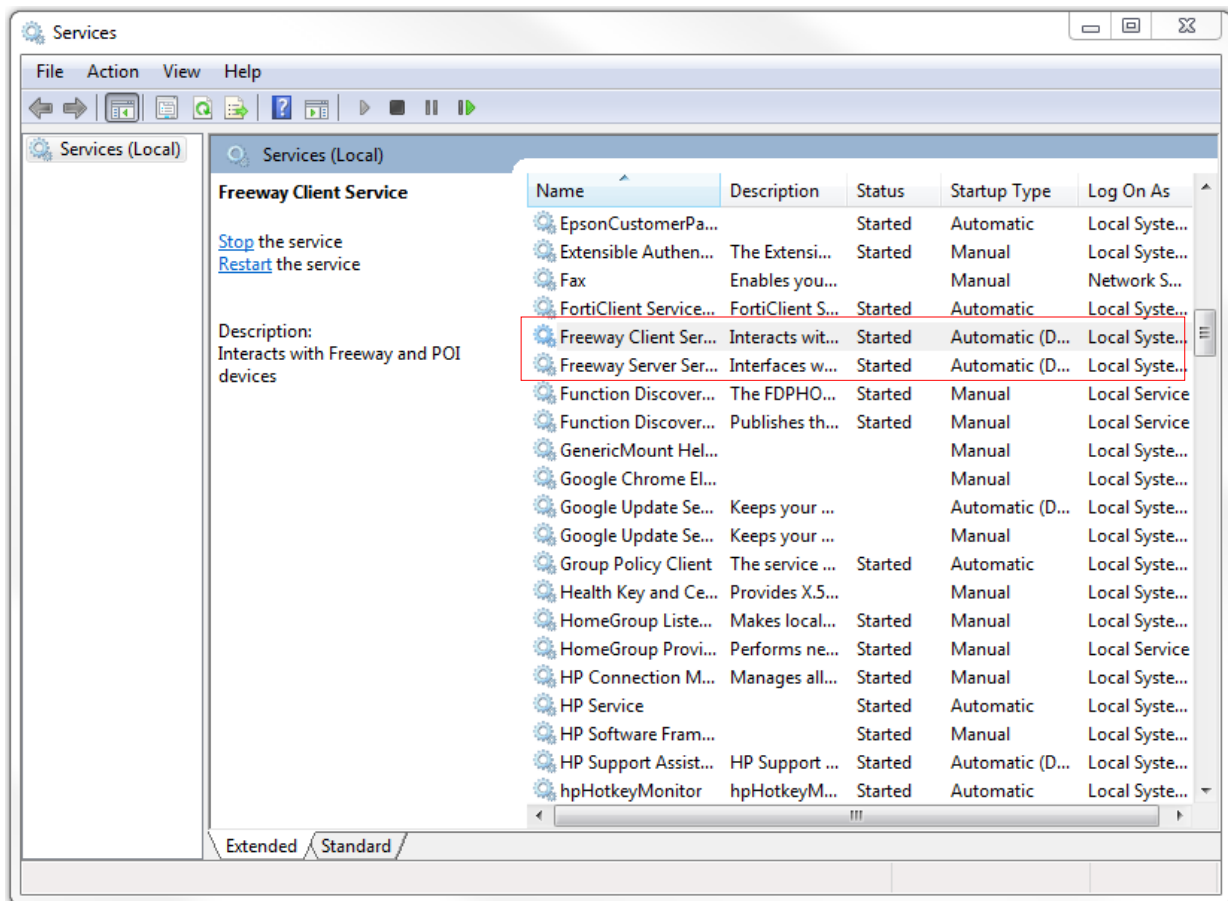
The point is that you can't merely see the "Secured by FreedomPay" on the Ingenico's screen and assume that the connection and configuration between Figaro, the Susanna and the Ingenico are right. If in doubt, reboot the Ingenico first by holding down the "Clear" and hyphen keys simultaneously.

Don't forget the prior discussion of "trash" in the store-id and/or terminal-id fields in the store configuration screen. If there are undisplayable characters in either field the authorization request will merely be ignored by Figaro and CounterPoint will just hang. If that's what you're experiencing, check that situation first.

## Restarting Figaro and Susanna

Oftentimes merely restarting Figaro and/or Susanna will correct an issue. Rebooting the machine they are installed on would certainly accomplish that, but restarting the machine will take a bit of time and could uncover new problems such as activate a Windows update that clobbered the Ingenico driver. Plus, the machine running Figaro is likely servicing other registers. You want to minimize the time that machine is offline and you definitely don't want to discover some new problem once it restarts.

It's better to merely restart the Figaro or Susanna services as opposed to rebooting the machine. You can do this by launching Windows Services applet and stopping, starting or restarting the service manually. The most ubiquitous way to do this (works for all versions of Windows) is to press the "Windows" key and enter "Services" (do I need to say not to type the quotes ?) in the resulting search prompt. One of the offerings will be "Services" with a set of gears as the icon. Select that. After a few moments you'll be presented a dialogue listing the services installed on the machine. Note that they might be installed, but that doesn't mean they are running. No doubt many are not. Go down the list looking for "Freeway Client Service" and/or "Freeway Server Service". If you have Figaro and Susanna installed on the same machine (as I do) they'll be side-by-side. Otherwise you'll just have one or the other. If you have neither then – Verne – there's your problem ! On my machine that looks like this.



You might notice both (as are others) are set to "Automatic (Delayed Start)". This setting prevents them from starting as soon as the machine starts up. This helps reduce processor load during startup.

The upper-left corner of the screen offers the options “Stop” and “Restart”. Since the selected service is running those are your options. If the service wasn’t running you’d be offered the single option “Start” to start the service.

You can click either. I prefer to stop the service, wait a couple seconds then start the service in separate motions. I could simply click Restart and get the same ultimate effect – unless there’s a problem going on. If that’s the case the process just hangs and you don’t know if it’s hanging shutting down or starting up. You end having to reboot the machine and, if you’re having a problem, do it again. I find it more prudent to just click Stop then Start.

An easier method is to use the scripts I provide with the Windows-based CP-EMV installation set. If you look in the Datafiles directory in CP-EMV’s installation set you’ll find 6 batch files: StartFigaro.bat, StopFigaro.bat, RestartFigaro.bat, StartSusanna.bat, StopSusanna.bat and RestartSusanna.bat. I bet you can figure out what they do by their names. You can copy one or all of these files to the target machine’s desktop and simply run them from there as needed. The one caveat is they must be run as an administrator. So rather than just double-clicking them you need to single-right click then single-left click “Run as administrator”. Windows will ask you if you want to run this program. Answer “Yup”. The batch will then run. It displays what’s going on as it runs including any errors it encounters.

Normally, each of these scripts displays “Press any key to continue...” as it completes. This pause allows you to see the results of the process before the window closes. But if you were to use one of these scripts in a larger script or as part of a scheduled job you wouldn’t want that last message to pop up. To suppress the “Press any key to continue...” prompt, define the environment variable “FCC\_Msg\_Pause” with the value “N”.

I don’t bundle these scripts with Linux installations. But I’ll email them to you upon request.

## **Log Files**

Figaro and Susanna both create/append to their own separate log files whenever they start, process a transaction, detect a Susanna or Ingenico joining the system, or stop. The default path for these log files is "C:\ProgramData\FreedomPay\Freeway Commerce Connect\log ". Each service has a configuration parameter that can be changed to put the log files wherever you want.

There's a lot of information in the logs file. They should be the first place you look when you're having problems. Almost all activity in Figaro and Susanna is logged. If you can't find the problem it might be because the problem isn't on the CounterPoint/Figaro/Susanna side of the equation. If it is and you can't determine the reason from the log files contact your FreedomPay support person, email them a detailed description of the problem of what you're seeing and attach the log files.

### **Figaro Log File**

Figaro's log files are named FCC\_Server\_YYYY-MM-DD\_#.xml where "YYYY-MM-DD" indicates the date and "#" is a segment number. Depending on the volume of traffic Figaro's log file may grow in size to be difficult to open using typical editors. The default segmentation size is one million bytes. If that proves to be too large you can take advantage of this "secret" Figaro configuration file parameter.

```
<add key="logMaxFileLength" value="500000"/>
```

The value "500000" directs Figaro to limit its log files to 500,000 bytes. That will make the files smaller and easier (or possible) to open, but it may lead to more log file to open when investigating problems.

Note that the log file's extension is ".xml". Figaro's log files are plain text files, but they don't read like one. They are formatted in XML which makes them a little harder to read, but they have the information you'll need. You can view it using pretty much any editor or NotePad. We suggest that if you intend to view the log file while it is processing transactions you need to ensure you use a program that doesn't lock the file while it is being accessed. Otherwise Figaro might start experiencing errors as it attempts to update the log file that your editor is holding locked. A good solution is to copy the log file to a different name and edit the copy. That prevents ever locking the original log file.

Here's an abbreviated sample of a swiped transaction in a Figaro log file. To keep it brief we've terminated lines that would have wrapped with "...".

```
<LogEntry type="NvpTransaction" timeStamp="2016-05-05 00:33:34.091" ProcessorTime=... >
```

```
<PosRequest timeStamp="2016-05-05 00:33:34.091">
  <POSRequest>
    <RequestType>Sale</RequestType>
    <ChargeAmount>120.34</ChargeAmount>
    <TaxAmount>10.23</TaxAmount>
    <AllowPartial>Y</AllowPartial>
    <ClientEnvironment>FCCTestClient 3.1.0.13</ClientEnvironment>
    <StoreId>1414424019</StoreId>
    <TerminalId>2414355011</TerminalId>
    <WorkstationId>1001</WorkstationId>
    <FloorLimit>50</FloorLimit>
    <MerchantReferenceCode>E92E84B777E94746</MerchantReferenceCode>
    <ClerkId>EGE</ClerkId>
    <InvoiceNumber>247</InvoiceNumber>
    <Offline>>false</Offline>
    <EnableAVS>Y</EnableAVS>
    <RegisterNumber>1</RegisterNumber>
    <TokenDynExp>>false</TokenDynExp>
  </POSRequest>
</PosRequest>
```

Authorization request  
from CounterPoint  
(might appear twice)

```
<ClientRequest timeStamp="2016-05-05 00:33:34.291">
```

```
<TransactionRequest>
  <StoreId>1414424019</StoreId>
  <TerminalId>2414355011</TerminalId>
  <ClerkId>EGE</ClerkId>
  <MerchantReferenceCode>E92E84B777E94746</MerchantReferenceCode>
  <InvoiceNumber>247</InvoiceNumber>
  <TransactionAmount>120.34</TransactionAmount>
  <AllowPartial>Y</AllowPartial>
  <TaxAmount>10.23</TaxAmount>
  <TipAmount>0</TipAmount>
  <RegisterNumber>1</RegisterNumber>
  <TransactionType>Purchase</TransactionType>
  <Options>None</Options>
  <PaymentDataSource>Card</PaymentDataSource>
  <TransactionMode>SaleMode</TransactionMode>
  <UseDcc>>false</UseDcc>
  <DCCInquiryOnly>>false</DCCInquiryOnly>
  <ClientApplication>FreewayIntegrationService</ClientApplication>
  <ClientApplicationVersion>3.1.0.13</ClientApplicationVersion>
  <ClientEnvironment>FCCTestClient 3.1.0.13</ClientEnvironment>
  <EnableAVS>Y</EnableAVS>
  <CreateTokenOnly>>false</CreateTokenOnly>
  <OriginalAmount>120.34</OriginalAmount>
  <TransactionTime>5/5/2016 12:33:34 AM</TransactionTime>
  <IsRefund>>false</IsRefund>
  <ForceOffline>>false</ForceOffline>
  <UpdateToken>>false</UpdateToken>
  <FloorLimit>50</FloorLimit>
  <Offline>>false</Offline>
  <GetSignature>>false</GetSignature>
  <ForceDeviceUpdate>>false</ForceDeviceUpdate>
  <TokenDynExp>>false</TokenDynExp>
  <IsToken>>false</IsToken>
</TransactionRequest>
</ClientRequest>
```

Authorization request  
sent to Susanna

```

<SusannaResponse timeStamp="2016-05-05 00:33:49.515">
  <TransactionResponse>
    <Decision>A</Decision>
    <ErrorCode>100</ErrorCode>
    <Message>APPROVED</Message>
    <RequestId>01Z6EDM15F01U611DHCIEVEVOFMGHINW</RequestId>
    <ApprovedAmount>120.34</ApprovedAmount>
    <MaskedCardNumber>371449XXXXX8431</MaskedCardNumber>
    <ApprovalCode>491949</ApprovalCode>
    <CardTypeCode>0</CardTypeCode>
    <ExpiryDate>0518</ExpiryDate>
    <DccElected>>false</DccElected>
    <authAndCaptureOnly>>true</authAndCaptureOnly>
    <SigCaptured>>true</SigCaptured>
    <EntryMode>swiped</EntryMode>
    <Signature>iVBORw0KGgoAAAANSUgAAALwAAs4c6QAA... </Signature>
    <RequestType>Purchase</RequestType>
    <NameOnCard>IVESTER/JAMIE </NameOnCard>
    <CardData>V=2|CT=credit|ET=onguard|MT=ingenico|EM...</CardData>
  </TransactionResponse>
</SusannaResponse>

```

Response from  
Susanna

```

<PosResponse timeStamp="2016-05-05 00:33:50.376">
  <POSResponse>
    <LaneId>0</LaneId>
    <Decision>A</Decision>
    <ErrorCode>100</ErrorCode>
    <Message>APPROVED</Message>
    <TransactionId>0000000000000000024</TransactionId>
    <RequestId>01Z6EDM15F01U611DHCIEVEVOFMGHINW</RequestId>
    <AccountBalance p2:nil="true" xmlns:p2="http://www.w3.org..." />
    <ApprovedAmount>120.34</ApprovedAmount>
    <TipAmount>0</TipAmount>
    <ApprovalCode>491949</ApprovalCode>
    <MaskedCardNumber>371449XXXXX8431</MaskedCardNumber>
    <CardType>credit</CardType>
    <NameOnCard>IVESTER/JAMIE </NameOnCard>
    <IssuerName>AMEX</IssuerName>
    <ExpiryDate>0518</ExpiryDate>
    <MerchantReferenceCode>E92E84B777E94746</MerchantReferenceCode>
    <Signature>iVBORw0KGgoAAAANSUgAAALwAAABMCAYAAAQAA...</Signature>
    <DCCAccepted>>false</DCCAccepted>
    <EntryMode>swiped</EntryMode>
  </POSResponse>
</PosResponse>
</LogEntry>

```

Response to  
CounterPoint  
(might appear twice)



## Susanna Log File

Susanna's log files are named FCC-client- *yyyy-mm-dd*.log where "*yyyy-mm-dd*" indicates the date. Susanna's log files are plain text files. You can view it using pretty much any editor or NotePad. We suggest that if you intend to view the log file while it is processing transactions you need to ensure you use a program that doesn't lock the file while it is being accessed. Otherwise Susanna might start experiencing errors as it attempts to update the log file that your editor is holding locked. A good solution is to copy the log file to a different name and edit the copy. That prevents ever locking the original log file.

Susanna's log files are only slightly easier to read than Figaro's. The reason is because they're loaded with geeky bits of information that would overwhelm you if you attempted to understand everything you see. Most of the lines are associated with communication with the Ingenico. You can ignore those and focus on the few lines related to the transaction coming in and going back to Figaro. Every line is tagged with the date and time of the entry. A transaction coming into Susanna always starts with a line "Received 4 bytes from *ip-address:3391*". The "*ip-address*" is going to be your Figaro's IP address. Around 8 lines below that you'll find a line that starts "Q:". That's the incoming authorization request. You should be able to see that it's a list of fields separated by the string "[FS]". Drop many lines down and you'll find a line that starts with "A:". That's the response string being sent back to Figaro. It's also a list of fields separated by the string "[FS]".

That's all you need to know for debugging transactions. There's a lot more information there surrounding connections to Figaro, the Ingenico and the gateway, but you might be getting into the weeds there. When Susanna starts you'll find the message "Service *version#* starting" (or something like that). Check that the version number is correct whenever you look at a log file since running the wrong version can lead to many strange results. When Susanna is shut down you'll see "Service stopping..." in the log file.

When Susanna attempts to connect to Figaro you should see the following lines although they may be separated by other log file entries.

```
[2016-04-27 03:51:27] GetMetafactory has a workstationID of : ED-8570P
[2016-04-27 03:51:27] Attempting to connect to server with workstationID of : 1001
[2016-04-27 03:51:27] Connecting to remote server
[2016-04-27 03:51:27] NetworkConnectionFactory: NetworkConnectionFactory: server 127.0.0.1:3391
[2016-04-27 03:51:27] NetworkConnectionFactory: Beginning connection to 127.0.0.1:3391
[2016-04-27 03:51:27] NetworkConnectionFactory: Connection to 127.0.0.1:3391 succeeded!
```

Replace "1001" with your workstation-id and "127.0.0.1" with your Figaro's IP address.

When Susanna attaches to its Ingenico you'll see a lot more lines. You can ignore most of them. You looking for lines that start and end like this.

```
[2016-04-27 03:51:26] Trying to attach to RBA device: \\.\libusb0-0001
[2016-04-27 03:51:26] Tx: 00.0000
[2016-04-27 03:51:26] Got ACK
[2016-04-27 03:51:28] [Lane 0] Tx: 28.900004121
(a whole bunch of lines skipped here)
[2016-04-27 03:51:28] [Lane 0] Tx: 70.TPROMPT311,311
[2016-04-27 03:51:29] [Lane 0] Got ACK
[2016-04-27 03:51:29] [Lane 0] SetStateIdle()
[2016-04-27 03:51:29] [Lane 0] Device attached
```

There are many detail levels of Susanna logging reporting. Logging level is controlled by a setting in Susanna's configuration file (FreewayClientService.exe.config).

```
<add key="LogLevel" value="Debug4" />
```

Possible values from the most minimum to the most verbose is "Critical", "Error", "Warning", "Information", "Verbose", "Debug1", "Debug2", "Debug3" and "Debug4". The default value is the most verbose – "Debug4". That's great if you're having problems and you probably want to keep it cranked up like that when you first start using CP-EMV. But over time, as you find you're not having problems you may decide to turn the setting down to a sane level. On the other hand, FreedomPay has done a good job in automating the zipping and ultimately the deleting old log files. So it's not like they'll continuously hog more and more disk space over time. And the time you want the extra detail is when you find you have a problem – not the next time you can replicate the problem. So if you can spare the disk space we'd suggest you leave the logLevel parameter at "Debug4".

Remember, whenever you make a change to Susanna's configuration file you need to restart the service before the changes will take effect.

## **Event Viewer Messages**

Figaro and the Susanna both write entries to the Windows Event Viewer when they start and stop. Messages from Figaro are labeled "FigaroSvc". Messages from the Susanna are labeled "FCCSvc". Frankly they don't help much since they only indicate whether the service started or stopped. Neither errors nor status information are written to the Event Viewer.

Figaro's configuration file has the following line in it.

```
<add key="eventLogName" value="FCCSvc"/>
```

That's the label used for Susanna's messages. Susanna doesn't have a similar configuration file setting. Changing the value of Figaro's "eventLogName" has no effect in the Event Viewer so don't be confused. Messages from Figaro are labeled "FigaroSvc". Messages from the Susanna are labeled "FCCSvc".

## **Duplicate Charges**

From time to time duplicate charges occur. It was more common years ago, but it still happens occasionally. The reasons are difficult to trace. Typically something happens on the FreedomPay side of the equation that results in the customer swiping their card more than once. But CounterPoint only ever receives the results of the last swipe. So CounterPoint only has the history for one of the charges.

As mentioned in [System>View>EMV Transaction history](#) one has the ability to void a transaction. As noted then, that is probably not the best solution. It leaves CounterPoint with an imbalance because the ticket history for the same is there, but no matching payment. The best solution is to void the first authorization – the one that CounterPoint didn't receive. If you get on FreedomPay's Enterprise Portal you can see both authorizations and their auth codes. Void (or perform a return against) the authorization whose auth code you don't have in CounterPoint. That way CounterPoint and FreedomPay match going forward.

## Contact Information

### **Infinity Software Solutions, Inc.**

1414 19<sup>th</sup> Street

Palm Harbor, FL 34683

CounterPoint Sales and Support: Ed Sills 727-785-8563 or email at [infinity\\_software@verizon.net](mailto:infinity_software@verizon.net)

### **FreedomPay Inc.**

FMC Tower at Cira Centre South

2929 Walnut Street · Floor 14

Philadelphia, PA 19104

Sales: Ed Learned at 763-515-4568 or email at [Ed.Learned@freedompay.com](mailto:Ed.Learned@freedompay.com)

Support: 888-495-2446 or email at [techsupport@freedompay.com](mailto:techsupport@freedompay.com)

General: 888-495-0222

Fax: 610-902-9001

# Sample Reports

## Customers>Reports>Customer cards

## Customers>Purge>Customer cards

### By Parameter

=====  
Date 99/99/99 Time 99:99:99 Camptown Hardware, Inc. Report# 9999 User XXX Page 9999

XX  
[ CUSTOMER CARDS LIST BY PARAMETERS ]  
[ PURGE CUSTOMER CARDS BY PARAMETERS ]

Customer-# range: XXXXXXXXXXXX to XXXXXXXXXXXX  
Card exp date range: mmyyxxxxxx to mmyyxxxx  
Cards to include: XXXXXXXXXXXX  
Token exp date range: 99/99/99xx to 99/99/99  
Include suspicious ? : XXXXXXX Export ? : Xxx " \* " To the right of the token expiration date indicates a token is present

Customer-#	Customer-name	Customer-phone	-----CustomerCardInfo-----				-----Card-----		Token	
			#	Card-name	Exp	Pay-code	Card-number	ExpDat	ExpDat	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
Rec#: 999,999 Reason: XXX										
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
Rec#: 999,999 Reason: XXX										
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	99/99	99/99/99*	

ZZZ,ZZ9 entries XXXXXXX  
[purged ]  
[printed]

# By Position

Date 99/99/99 Time 99:99:99

Camptown Hardware, Inc.

Report# 9999 User XXX Page 9999

XX  
 [ C U S T O M E R C A R D S L I S T B Y P O S I T I O N ]  
 [ P U R G E C U S T O M E R C A R D S B Y P O S I T I O N ]

Customer-# range: XXXXXXXXXXXX to XXXXXXXXXXXX

Card-name	Exp	Pay-code	Include?	Card-name	Exp	Pay-code	Include?		
XXXXXXXXXX	N	99	XXXXXXXXXX	X	XXXXXXXXXX	N	99	XXXXXXXXXX	X
XXXXXXXXXX	N	99	XXXXXXXXXX	X	XXXXXXXXXX	N	99	XXXXXXXXXX	X
XXXXXXXXXX	N	99	XXXXXXXXXX	X	XXXXXXXXXX	N	99	XXXXXXXXXX	X
XXXXXXXXXX	N	99	XXXXXXXXXX	X	XXXXXXXXXX	N	99	XXXXXXXXXX	X

Include suspicious ? : XXXXXX Export ? : Xxx "\*" To the right of the token expiration date indicates a token is present

Customer-#	Customer-name	Customer-phone	-----CustomerCardInfo-----				-----Card-----		Token	
			#	Card-name	Exp	Pay-code	Card-number	ExpDat	ExpDat	
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
Rec#: 999,999 Reason: XXX										
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
Rec#: 999,999 Reason: XXX										
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*
XXXXXXXXXX	XX	XX	9	XXXXXXXXXX	N	99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99/99	99/99/99*

ZZZ,ZZ9 entries XXXXXX  
 [purged ]  
 [printed]

# Customers>Cash Receipts>Edit list

Date 99/99/99 Time 99:99:99

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Report# XXXX User XXX Page 9999

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
 [ C A S H R E C E I P T S E D I T L I S T ]  
 [ C A S H R E C E I P T S R E G I S T E R ]

Batch user ID: XXXXXXXXXXXX  
 "\*" beside authorization code indicates a voice authorization.

Customer-#	Name	Pay-code	Check-#	Date	Amount-recvd	Reference	Wrtoff/non-AR-acct
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99XXXXXXXXXX	XXXXXXXXXX	99/99/99	999,999,999.99-	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99999999-99999999
		Card-#:	XXXXXXXXXXXX	Auth:	XXXXXX	Pay code distribution account:	99999999-99999999
		Ref-Id:	Z9/99/99	99:99:99	Sale	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	
		Ref-Id:	Z9/99/99	99:99:99	Retn	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99XXXXXXXXXX	XXXXXXXXXX	99/99/99	999,999,999.99-	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99999999-99999999
		Card-#:	XXXXXXXXXXXX	Auth:	XXXXXX	Pay code distribution account:	99999999-99999999
		Ref-Id:	Z9/99/99	99:99:99	Sale	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	
		Ref-Id:	Z9/99/99	99:99:99	Retn	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99XXXXXXXXXX	XXXXXXXXXX	99/99/99	999,999,999.99-	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99999999-99999999
		Card-#:	XXXXXXXXXXXX	Auth:	XXXXXX	Pay code distribution account:	99999999-99999999
		Ref-Id:	Z9/99/99	99:99:99	Sale	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	
		Ref-Id:	Z9/99/99	99:99:99	Retn	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	

ZZZZZ9 cash receipt for batch user ID XXXXXXXXXXXX Total received: 999,999,999.99-

ZZZZZ9 non-AR recpt for batch user ID XXXXXXXXXXXX Total non-AR: 999,999,999.99-

Total cash receipts for batch user ID XXXXXXXXXXXX ZZZZZ9 99XXXXXXXXXX 999,999,999.99-  
 Total received: 999,999,999.99-

# Customers>Reports>Open item detail

Date 99/99/99 Time 99:99:99

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

Report# XXXX User XXX Page 9999

## O P E N I T E M D E T A I L

Customer range: XXXXXXXXXXXX to XXXXXXXXXXXX  
Date range: Z9/99/99xx to Z9/99/99

Document types: I = Invoice P = Payment C = Credit memo D = Debit memo F = Finance charge B = Balance forward

Doc-#	Typ	Date	Apply-to	Due-date	Str	Sales-rep	P.O.#	Order-#	Terms	Amount	Other-amount	Balance
-------	-----	------	----------	----------	-----	-----------	-------	---------	-------	--------	--------------	---------

Customer: XXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXX  
=====

XXXXXXXXXX	X	Z9/99/99	XXXXXXXXXX	99/99/99	999	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-			
										EMV-ReqId: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	EMV: X	
XXXXXXXXXX	X	Z9/99/99	XXXXXXXXXX	99/99/99	999	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-			
										EMV-ReqId: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	EMV: X	
XXXXXXXXXX	X	Z9/99/99	XXXXXXXXXX	99/99/99	999	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-			
										EMV-ReqId: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	EMV: X	

Customer totals: ZZZ,ZZ9 open items ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-  
(Customer balance: ZZZ,ZZZ,ZZZ.99-) Open item totals:ZZZ,ZZZ,ZZZ.99-

Grand totals: ZZZ,ZZ9 customers ZZZ,ZZ9 open items ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-  
(Total customer balance: ZZZ,ZZZ,ZZZ.99-) Open item totals:ZZZ,ZZZ,ZZZ.99-





[XXX]
[EMV TRANSACTION HISTORY REPORT]
[PURGE EMV TRANSACTION HISTORY]

Store-#: 99zxx
Date range: 99/99/99xx to 99/99/99
Record type: XXXXXXXXXXXXXXXXXXXX

MICRO FORMAT
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
[ BY DATE ]
[ BY STORE#/DATE ]

Transaction type: XXXXXXXXXXXXXXXXXXXX

Table with columns: Store, Date, Time, Record-id, TrxTyp, AuthCod, RequestAmt, ApproveAmt. Contains 10 rows of transaction data.

ZZZ,ZZZ,ZZ9 Totals records

Totals by record type:

ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX

Totals by transaction type:

ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX
ZZZ,ZZZ,ZZ9 XXXXXXXXXXXXXXXXXXXXXXXX

# Summary Format

Date 99/99/99 Time 99:99:99

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Report# XXXX User XXX Page 9999

[XX]  
 [E M V T R A N S A C T I O N H I S T O R Y R E P O R T]  
 [ P U R G E E M V T R A N S A C T I O N H I S T O R Y ]

Store-#: 99zxx  
 Date range: 99/99/99xx to 99/99/99  
 Record type: XXXXXXXXXXXXXXXXXXXX

SUMMARY (1-LINE)

XXXXXXXXXXXXXXXXXXXXXXXXXXXX  
 [ BY DATE ]  
 [ BY STORE#/DATE ]  
 [BY STORE#/RECORD TYPE]

Transaction type: XXXXXXXXXXXXXXXXXXXX

TrxTyp	NameOnCard	CardType	MaskedCardNo	ExpDt	Off	AuthCod	RequestAmt	ApproveAmt
[----- Two lines appear here providing the details of the CounterPoint -----] [----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
[----- Two lines appear here providing the details of the CounterPoint -----] [----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
[----- Two lines appear here providing the details of the CounterPoint -----] [----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	

**Brief Format**

=====  
 Date 99/99/99 Time 99:99:99 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX Report# XXXX User XXX Page 9999

[XX]  
 [EMV TRANSACTION HISTORY REPORT]  
 [PURGE EMV TRANSACTION HISTORY]

Store-#: 99zxx  
 Date range: 99/99/99xx to 99/99/99  
 Record type: XXXXXXXXXXXXXXXXX

BRIEF (3-LINES)  
 XXXXXXXXXXXXXXXXXXXXXXXX  
 [ BY DATE ]  
 [ BY STORE#/DATE ]  
 [BY STORE#/RECORD TYPE]

Transaction type: XXXXXXXXXXXXXXXXX

TrxTyp	NameOnCard	CardType	MaskedCardNo	ExpDt	Off	AuthCod	RequestAmt	ApproveAmt
RecTyp	RequestId	EntryMode	Issuer	CVM	Dec	ErrCod	AccountBal	PartialAmt
Str	MerchantRefCod		Transaction-id	DCC	Sig	LaneId	FloorLimit	TipAmt
[----- Two lines appear here providing the details of the CounterPoint -----]								
[----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXX	X	ZZZZZ9	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
ZZ9	XXXXXXXXXXXXXXXXXXXX		XXXXXXXXXXXXXXXXXXXX	X	X	ZZZZZZZZ9	ZZZ,ZZZ,ZZZ.99 ZZZ,ZZZ,ZZZ.99-	
[----- Two lines appear here providing the details of the CounterPoint -----]								
[----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXX	X	ZZZZZ9	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
ZZ9	XXXXXXXXXXXXXXXXXXXX		XXXXXXXXXXXXXXXXXXXX	X	X	ZZZZZZZZ9	ZZZ,ZZZ,ZZZ.99 ZZZ,ZZZ,ZZZ.99-	
[----- Two lines appear here providing the details of the CounterPoint -----]								
[----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXX	X	ZZZZZ9	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
ZZ9	XXXXXXXXXXXXXXXXXXXX		XXXXXXXXXXXXXXXXXXXX	X	X	ZZZZZZZZ9	ZZZ,ZZZ,ZZZ.99 ZZZ,ZZZ,ZZZ.99-	
[----- Two lines appear here providing the details of the CounterPoint -----]								
[----- transaction associated with the EMV activity (see below) -----]								
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXX	X	ZZZZZ9	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
ZZ9	XXXXXXXXXXXXXXXXXXXX		XXXXXXXXXXXXXXXXXXXX	X	X	ZZZZZZZZ9	ZZZ,ZZZ,ZZZ.99 ZZZ,ZZZ,ZZZ.99-	

# Full Format

=====  
 Date 99/99/99 Time 99:99:99 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX Report# XXXX User XXX Page 9999

[XX]  
 [E M V T R A N S A C T I O N H I S T O R Y R E P O R T]  
 [ P U R G E E M V T R A N S A C T I O N H I S T O R Y ]

Store-#: 99zxx  
 Date range: 99/99/99xx to 99/99/99  
 Record type: XXXXXXXXXXXXXXXXXXXX

FULL (EMV TAG DATA)  
 XXXXXXXXXXXXXXXXXXXXXXXX  
 [ BY DATE ]  
 [ BY STORE#/DATE ]  
 [BY STORE#/RECORD TYPE]

Transaction type: XXXXXXXXXXXXXXXXXXXX

TrxTyp	NameOnCard	CardType	MaskedCardNo	ExpDt	Off	AuthCod	RequestAmt	ApproveAmt
RecTyp	RequestId	EntryMode	Issuer	CVM	Dec	ErrCod	AccountBal	PartialAmt
Str	MerchantRefCod		Transaction-id	DCC	Sig	LaneId	FloorLimit	TipAmt

[----- Two lines appear here providing the details of the CounterPoint -----]  
 [----- transaction associated with the EMV activity (see below) -----]

Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXX	XXXXX	X	ZZZZZ9	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
ZZ9	XXXXXXXXXXXXXXXXXXXXXXXX		XXXXXXXXXXXXXXXXXXXXXXXX	X	X	ZZZZZZZZZ9	ZZZ,ZZZ,ZZZ.99 ZZZ,ZZZ,ZZZ.99-	
	EMV-TAG-DATA 50: XXXXXXXXXXXXXXXXXXXXXXX	9F03: XXXXXXXXXXX		9F27: XXXXX			ARC: XXXXX	
	5F2A: XXXXX	9F07: XXXXX		9F34: XXXXXXXXXXX			IAD: XXXXXXXXXXXXXXX	
	5F34: XXXXX	9F0D: XXXXXXXXXXX		9F36: XXXXX			TSI: XXXXX	
	82: XXXXX	9F0E: XXXXXXXXXXX		9F37: XXXXXXXXXXX			TVR: XXXXXXXXXXX	
	95: XXXXXXXXXXX	9F0F: XXXXXXXXXXX		DF03: XXXXXXXXXXXXXXX			TAC-DEFAULT: XXXXXXXXXXX	
	9A: XXXXXXXXXXX	9F12: XXXXXXXXXXXXXXX		DF04: XXXXXXXXXXX			TAC-DENIAL: XXXXXXXXXXX	
	9C: XXXXXXXXXXX	9F1A: XXXXX		DF05: XXXXXXXXXXX			TAC-ONLINE: XXXXXXXXXXX	
	9F02: XXXXXXXXXXX	9F26: XXXXXXXXXXXXXXX		AID: XXXXXXXXXXXXXXX				

[----- Two lines appear here providing the details of the CounterPoint -----]  
 [----- transaction associated with the EMV activity (see below) -----]

Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXX	XX/XX	X	XXXXXXXX	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
Xxxxxx	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXX	XXXXX	X	ZZZZZ9	ZZZ,ZZZ,ZZZ.99-ZZZ,ZZZ,ZZZ.99-	
ZZ9	XXXXXXXXXXXXXXXXXXXXXXXX		XXXXXXXXXXXXXXXXXXXXXXXX	X	X	ZZZZZZZZZ9	ZZZ,ZZZ,ZZZ.99 ZZZ,ZZZ,ZZZ.99-	
	EMV-TAG-DATA 50: XXXXXXXXXXXXXXXXXXXXXXX	9F03: XXXXXXXXXXX		9F27: XXXXX			ARC: XXXXX	
	5F2A: XXXXX	9F07: XXXXX		9F34: XXXXXXXXXXX			IAD: XXXXXXXXXXXXXXX	
	5F34: XXXXX	9F0D: XXXXXXXXXXX		9F36: XXXXX			TSI: XXXXX	
	82: XXXXX	9F0E: XXXXXXXXXXX		9F37: XXXXXXXXXXX			TVR: XXXXXXXXXXX	
	95: XXXXXXXXXXX	9F0F: XXXXXXXXXXX		DF03: XXXXXXXXXXXXXXX			TAC-DEFAULT: XXXXXXXXXXX	
	9A: XXXXXXXXXXX	9F12: XXXXXXXXXXXXXXX		DF04: XXXXXXXXXXX			TAC-DENIAL: XXXXXXXXXXX	
	9C: XXXXXXXXXXX	9F1A: XXXXX		DF05: XXXXXXXXXXX			TAC-ONLINE: XXXXXXXXXXX	
	9F02: XXXXXXXXXXX	9F26: XXXXXXXXXXXXXXX		AID: XXXXXXXXXXXXXXX				

The lines below detail how those two lines (referred to above) are formatted for each record type

A/R Cash Receipt Transactions

User-id	Customer	Customer-name	Seq	Date	Time
XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX	ZZZZ9	Z9/99/99	Z9:99:99

A/R Open/Closed Item Transactions

Customer	Customer-name	Date	Doc-#	Seq	Typ	Apply-to	Seq#	Date	Time
XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX	Z9/99/99	XXXXXXXXXX	Z99	X	XXXXXXXXXX	Z99	Z9/99/99	Z9:99:99

Ad Hoc Transactions

User-id	Customer-#	Customer-name	Reg
XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX	ZZ9 XXXXXXXXXXXXXXXXXXXXXXXXX

Non-A/R Customer Receipts

Customer	Customer-name	Date	Time
XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX	Z9/99/99	Z9:99:99

O/E Order Deposits

Type	Store	Order#	PayTyp	Date	Time	Customer	Customer-name
XXXXX	ZZ9	ZZZZZ9	XXXXXX	Z9/99/99	Z9:99:99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX

Pos Ticket Payments

Type	Reg	Ticket	Idx	Date	Time	Customer	Customer-name
XXXXXX	ZZ9	ZZZZZ9	Z9	Z9/99/99	Z9:99:99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX

Sales History Payments

Str	Ticket	Seq	Idx	Date	Time	Customer	Customer-name
ZZ9	ZZZZZ9	ZZ9	Z9	Z9/99/99	Z9:99:99	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXX

# System>Reports>EMV Reconciliation report

## Brief Format

-----  
Date 99/99/99 Time 99:99:99

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

Report# XXXX User XXX Page 9999

E M V R E C O N C I L I A T I O N R E P O R T

Store-#: 999xx

\*\*\*\*\* BRIEF FORMAT \*\*\*\*\*

Date range: 99/99/99xx to 99/99/99

Issuer: XXXXXXXXXXXXXXXXXXXX

Store-#: 999 XXXXXXXXXXXXXXXXXXXX

-----  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)  
Total for XXXXXXXXXXXXXXXXXXXX: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)

Total for store ZZ9: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)

Grand total: ZZZ,ZZZ,ZZZ.99- (in ZZZ,ZZZ,ZZ9 transactions)



**FileUtilities>Special>System>Mask credit cards**

Date 99/99/99 Time 99:99:99

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

Report# XXXX User XXX Page 9999

M A S K C R E D I T C A R D S

Cust-#-range: XXXXXXXXXXXX to XXXXXXXXXXXX \*\*\*\*\* MASKING CUSTOMER FILE \*\*\*\*\*  
Action to take: XXXXXXXXXXXX  
EMV Pay code used: 9. XXXXXXXXXXX (XXXX-XXXXXXXX)

\*\*\* Skipped ticket history file \*\*\*

Customer-#	Customer-name	-----Existing card----- Card-number	ExpDt	-----New token----- Token	ExpDate
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	*XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	Z9/99 Z9/99 Z9/99 Z9/99 Z9/99	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	99/99/99 99/99/99 99/99/99 99/99/99 99/99/99
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	*XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	Z9/99 Z9/99 Z9/99 Z9/99 Z9/99	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	99/99/99 99/99/99 99/99/99 99/99/99 99/99/99
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	*XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	Z9/99 Z9/99 Z9/99 Z9/99 Z9/99	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	99/99/99 99/99/99 99/99/99 99/99/99 99/99/99
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	*XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	Z9/99 Z9/99 Z9/99 Z9/99 Z9/99	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	99/99/99 99/99/99 99/99/99 99/99/99 99/99/99
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	*XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	Z9/99 Z9/99 Z9/99 Z9/99 Z9/99	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	99/99/99 99/99/99 99/99/99 99/99/99 99/99/99

Records read: ZZZ,ZZZ,ZZ9  
Records updated: ZZZ,ZZZ,ZZ9





# File Utilities>Special>System>Set CP-EMV fields

Date 99/99/99 Time 99:99:99

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Report# XXXX User XXX Page 9999

S E T G A T E W A Y T O U S E

[XXXXXXXXXXXXXXXXXXXXXXXXXXXX]  
 \*\*\* REPORT-ONLY \*\*\*  
 \*\*\* UPDATE \*\*\*

Str	Gateway	Ver	Store-id	Term-id	IP-address	Port	T/O	Sav sig	Floor limit
-----	---------	-----	----------	---------	------------	------	-----	---------	-------------

Parameters:

```

Credit card gateway      XXXXXXXXXXXX      (Override)
FCC Version number      XXX              (Override)

FreedomPay store-id     XXXXXXXXXXXXXXXX  (Override)
FreedomPay terminal-id  XXXXXXXXXXXXXXXX  (Override)

FCC Server IP address   XXXXXXXXXXXXXXXX  (Override)
FCC Server port         ZZ,ZZ9           (Override)
FCC Server timeout (seconds) ZZ9              (Override)

Store signatures locally ? XXX              (Override)
Floor limit             Z,ZZZ,ZZ9       (Override)
Operating mode          XXXXXXXXXXXX     (Override)
  
```

	Str	Gateway	Ver	Store-id	Term-id	IP-address	Port	T/O	Sav sig	Floor limit	Operating mode
(different)	ZZ9	XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
		XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
(different)	ZZ9	XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
		XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
(changed)	ZZ9	XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
		XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
(changed)	ZZ9	XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX
		XXXXXXXXXX	Xxxx	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	ZZZZ9	ZZ9	Xxx	Z,ZZZ,ZZ9	XXXXXXXXXX

**File Utilities>Special>Customers>Tokenize customer cards**

Date 99/99/99 Time 99:99:99

Camptown Hardware, Inc.

Report# 9999 User XXX Page 9999

T O K E N I Z E C U S T O M E R C A R D S

Cust-# range: XXXXXXXXXXXX to XXXXXXXXXXXX  
 Customer type: XXXXXXXXXXXX  
 Customer category: XXXXX XXXXXXXXXXXXXXXXXXXXXXXX  
 EMV Pay code used: XXXX XXXXXXXXXXXX  
 Delete tokenized cards ?: X

Customer-#	Customer-name	-----Existing card----- Card-number	ExpDt	-----New token----- Token	ExpDate	
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	*XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
		XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
		XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
		XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
		XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	Z9/99	XXXXXXXXXXXXXXXXXXXX	99/99/99	XXXXXXXXXXXXXXXXXXXX

Records read: ZZZ,ZZZ,ZZ9  
 Records updated: ZZZ,ZZZ,ZZ9

Date 99/99/99 Time 99:99:99

Camptown Hardware, Inc.

Report# 9999 User XXX Page 9999

ARCUSP.RPT

# File Utilities>Special>Customers>Set allow customer cards

## S E T C U S T O M E R A L L O W T O K E N S

Enable/disable tokens: Xxxxxxx [XXXXXXXXXXXXXXXXXXXXX]  
Customer-# range: XXXXXXXXXXXX to XXXXXXXXXXXX [\*\*\* REPORT-ONLY \*\*\*]  
Customer type: Xxxxxxxxx [ \*\*\* UPDATE \*\*\* ]  
Customer category: XXXXX XXXXXXXXXXXXXXXXXXXXXXXX

```
-----
```

Customer-#	Customer-name	Type	Cat
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXX	Xxxxxxxxx	XXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXX	Xxxxxxxxx	XXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXX	Xxxxxxxxx	XXXXX
XXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXX	Xxxxxxxxx	XXXXX

```
-----
```

ZZZ,ZZ9 customers xxxxxxx  
[listed]  
[updated]

**File Utilities>Special>Customers>Create tokens from history**

Date 99/99/99 Time 99:99:99

Camptown Hardware, Inc.

Report# 9999 User XXX Page 9999

C R E A T E C U S T O M E R C A R D S F R O M H I S T O R Y

Date-range: 99/99/99 to 99/99/99  
Cust-# range: XXXXXXXXXXXX to XXXXXXXXXXXX  
Customer type: XXXXXXXXXXXX  
Customer category: XXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXX  
EMV Pay code used: XXXX XXXXXXXXXXXX

-----  
KEY FIELD COMPOSITION BY RECORD TYPE  
-----

A = "A/R Cash receipt"	A-uuuuuuuuuu-cccccccccc-qqqqq
B = "A/R Open item"	B-cccccccccc-dddddddd-nnnnnnnnn-sss-y-aaaaaaaa-qqq
C = "A/R Closed item"	C-cccccccccc-dddddddd-nnnnnnnnn-sss-y-aaaaaaaa-qqq
N = "Non-A/R Cash rcpt"	N-cccccccccc
O = "O/E Order"	O-y-sss-ooooo-p
P = "POS Tkts/Ords"	P-y-rrr-tttttt-ii
L = "POS Tkt/Ord lines"	L-y-rrr-tttttt-l-mmmmm-qqqqq
S = "Hist Tkts/Ords"	S-sss-tttttt-qqq-ii
U = "Hist Tkt/Ord lines"	S-sss-tttttt-qqq-l-mmmmm-qqqqq
T = "Token"	T-cccccccccc

KEY FIELD CODE LEGEND  
-----

a = apply-to number  
c = customer-#  
d = transaction date  
i = payment index  
l = line type  
m = line number  
n = Document number  
o = order number  
p = payment type  
q = sequence-#  
r = register number  
s = store number  
t = ticket number  
u = user-id  
y = transaction type

Date 99/99/99 Time 99:99:99

Camptown Hardware, Inc.

Report# 9999 User XXX Page 9999

C R E A T E C U S T O M E R C A R D S F R O M H I S T O R Y

Date-range: 99/99/99xx to 99/99/99

