

DC EMV DECODED

Using DC EMV to build a complete closed loop payment system from card applets to terminal and EMV kernels to backend transaction switching running in Kubernetes on Azure.



Vicente Da Silva

V 1.2

DC EMV DECODED Copyright © 2018 by Vicente Da Silva. All Rights Reserved.

All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means including information storage and retrieval systems, without permission in writing from the author. The only exception is by a reviewer, who may quote short excerpts in a review.

1 Table of Contents

- 2 Overview 7
 - 2.1 Looking forward 11
 - 2.2 Acknowledgements 13
 - 2.3 Hardware Drivers for Card Readers 16
 - 2.4 EMV Kernels and Terminal Libraries 17
 - 2.5 EMV Card Applets..... 18
 - 2.5.1 HCE18
 - 2.6 The Closed Loop Payment Demo System..... 19
 - 2.6.1 Demo Client App19
 - 2.6.2 Demo Server Software19
 - 2.7 Tools Used 21
 - 2.7.1 Visual Studio.....21
 - 2.7.2 SQL Server21
 - 2.7.3 IntelliJ21
 - 2.7.4 NSwag22
- 3 EMV Core Libraries 23
 - 3.1 Hardware Drivers for Card Readers 24
 - 3.1.1 Windows (PC/SC)24
 - 3.1.2 Android (Built In).....24
 - 3.1.3 Android (ACS1255).....24
 - 3.1.4 Windows IOT (NXP OM5577).....24
 - 3.1.5 TPM25
 - 3.2 EMV Terminal Applications 26

3.2.1	EMV Terminal Configuration	26
3.2.2	Contact and Contactless Terminal Application.....	27
3.3	EMV Kernels	29
3.3.1	EMV Kernel Configuration	29
3.3.2	EMV Contactless Kernel 1 and 3	31
3.3.3	EMV Contactless Kernel 2	31
3.3.4	EMV Contact Kernel	31
3.3.5	EMV QR Codes	32
4	EMV Card Applets.....	33
4.1	PPSE Applet Features	33
4.2	DCEMV Payment Applet Features.....	33
4.3	HCE version of Card Applets.....	33
5	EMV Card Personalization Application	35
6	Demo Closed Loop Payment System	36
6.1	Demo Client App	36
6.1.1	Demo Client Server Proxy Generation with Swagger	37
6.2	Demo Server Application	38
6.2.1	Architecture	38
6.2.2	Installing the Database	38
6.2.3	Creating the Database	43
6.2.4	Configuring the Database	44
6.2.5	Running the Demo Server App	45
6.3	Online Approver Drivers.....	58
6.3.1	Auth Driver configuration	59
6.3.2	ContactlessDummyOnlineApprover	61

6.3.3	SPDH Driver	61
6.3.4	DCEMVServerOnlineApprover	61
6.3.5	SimulatedApprover	61
6.3.6	ISO8583 Driver	61
7	Using the System	62
7.1	Install Tools.....	62
7.1.1	Configure Windows.....	62
7.1.2	Docker for Windows	62
7.1.3	Visual Studio.....	63
7.1.4	IntelliJ	68
7.2	Implementing the Card Applets	71
7.2.1	In a Simulator	71
7.2.2	On a Physical Card.....	75
7.2.3	HCE on device	76
7.3	Run Server App Standalone.....	77
7.4	Run Demo App	79
7.4.1	Transactions with Cards or HCE	97
7.4.2	Transactions with QR Codes	100
7.5	Run Personalization App	103
7.5.1	Understanding Card Keys.....	103
7.5.2	Understand the Perso File	104
7.5.3	Perso the Card.....	106
7.5.4	Re-Perso the Card	112
7.6	Demo Application for EMV Contact and Contactless Kernels	
	113	

7.7	Running the apps on Android.....	117
7.8	Running the apps on the Raspberry Pi.....	117
8	Sample Terminal Application Log	120

2 Overview

Dead Coral EMV (DC EMV) is an implementation of the entire stack of software needed for a closed loop payment system using EMV standards. Why Dead Coral EMV? I decided that as part of releasing this software I would use the release as an opportunity to draw attention to one of many environmental issues facing us today and one that I feel strongly about. See www.chasingcoral.com for more information.

DC EMV includes many components. The project started as an implementation of a closed loop system using MiFare cards, however I decided to switch to EMV. EMV was a better fit as I could issue EMV cards to my customers, to use within the closed loop system and they could use their bank EMV cards to top up the funds available in their closed loop system account. I then implemented EMV contactless kernels 1, 2 and 3. A Terminal library was then built and used within the views of the Closed Loop Payment client app. A Server App was built that could run Standalone, in Docker or in Kubernetes in Azure.

I wanted the system to run on a wide range of hardware as possible. By using Xamarin I am able to have the client software run on Windows 10, Windows IOT, Android and iOS with a shared code base. EMV cards need card readers. I have tried to build a range of pluggable card drivers that allow the software to run on a wide range of hardware. It can run on Android tablets and phones with or without built-in NFC hardware. It can run on Windows via PS/SC compatible card readers of which there are many. It runs on a Raspberry Pi with an OM5577 card reader plugin board. This was one of the first drivers to be built. At the time there were no native drivers for card readers built into Windows IOT. I built an NCI I2C driver for the card and as far as I know this is one of the only ways to interface to an EMV card on the Raspberry Pi, there are some readers for NFC tags but not for ISO7816 contactless cards.

The server software is build using ASP.NET Core and implements OAuth2 for authentication between client and server. It runs on Docker, meaning it can run on any IaaS provider.

At this point the intention was to buy EMV cards, have them personalized and issue them to users. This proved difficult as card vendors are difficult to work with, offering little in terms of documentation but very eager to sell consulting fees. I decided to build my own EMV compatible card app. This Card Payment Applet and its companion PPSE Applet are EMV compatible with kernel 3. They always request online authentication via an ARQC cryptogram.

In order to validate the cryptogram an in-memory HSM was built to manage keys in a separate environment and to also verify the cryptogram. This in-memory HSM is simply a library and needs to be fleshed out into a standalone application offering security functions via web services or low-level socket comms.

Once you have built a card app, you need to load it onto a card. I then built a Card Personalization application that used the Global Platform standard. Conforming to this standard allows the Applets to run on cards from many different manufacturers. In the process of developing the applets I built a card emulator that listens for ADPU data via TCP/IP and feeds this into the card applet therefor making it easy to test and debug a card applet without needing any hardware. A companion card reader component, used in the client applications, was built that creates a virtual card reader and forwards any ADPU commands sent to it to the card emulator.

For the top up function an EMV contactless (or contact depending on the hardware used) card could be used, but these transactions would need to be sent to the issuing institution for authorization. There are a few different pluggable authentication modules to enable this. One of

these is a simple SPDH based auth driver. If you are using EMV test cards and know the master AC key, then there is another auth module, the DCEMV_SimulatedPaymentProvider that will generate an ARPC for the card being used. This auth module can also generate a pin change script providing the MAC and DEA master keys are known. This negates having to connect to an external system to get authorizations. I plan to build these backend components into a highly scalable transaction switch.

At this point the project had grown quite large. I decided the way forward would be to open source it, rather than create a closed loop payment system. Open sourcing it will give it the “wings” it needs to grow into something much more. I felt that for it to become a full EMV kernel solution it needed a contact kernel. This was one of the last components to be built before open sourcing the project.

Figure 1 shows a high-level view of the main software components of the system.

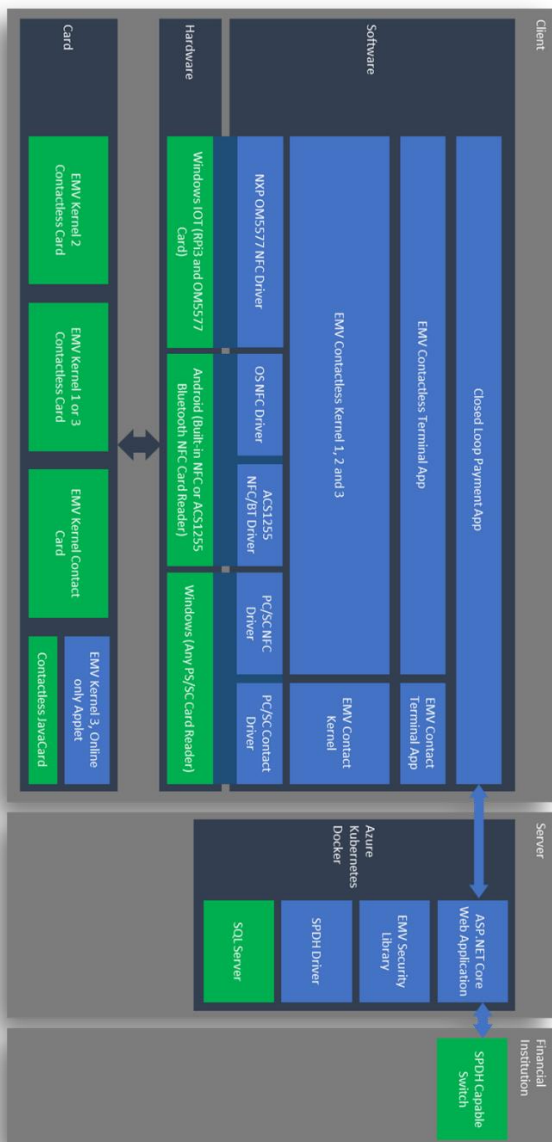


Figure 1

2.1 Looking forward

Features / Enhancements planned are as follows:

1. “Pre-certify” the kernels by using certified testing tools such as the test tool from UL along with an Astrex or MasterCard and Visa simulator. If you have access to these tools and want to contribute to the project then please contact me.
2. In order to use these kernels in a production environment where association cards are going to be accepted, the kernel needs to be certified on specific hardware. If there are any vendors making EMV level 1 certified hardware and who are interested in integrating this kernel onto their devices, please contact me.
3. A magstripe kernel should be added to the code base, for legacy cards, if there is anyone who has had this kind of experience and wishes to contribute to the project, please contact me.
4. Performance enhancements to speed up transaction execution time.
5. Enhance the card applets to support both contact and contactless applications. With these a test tool can be created that implements the test pack/cases (for each card association) that a terminal vendor would need to certify their terminal against in order to get certification. This should not be that hard as the card state engine is a lot simpler than the kernels. Alternatively, a company like UL could enhance their test tool with an API that can be called by a test application to load cards rather than having to physically select the card to be loaded via the UI.
6. Develop an acquirer/issuer switch capable of routing transactions between systems using different protocols as well

as approving / declining “on us” transactions. All the cryptographic requirements for this switch are already implemented in the Simulated Payment Provider module.

7. Improve testability to the point where a full regression test can be done in completely automated way, using simulated cards and the switch mentioned above. This automated testing would be able to check card logs, terminal logs and switch logs and report on discrepancies across the entire transaction flow for all test cases in a matter of minutes.
8. Certify physical cards for use with the applets. Any card manufacturers interested in having their cards certified for use with the DC EMV payment applets should contact me. I would like to be able to provide users of the DC EMV framework a list of cards certified to work with the applets. This will negate them having to find compatible cards themselves and will shorten their development cycle.
9. Enhance the HSM module to do tokenization of cards. This simply requires adding an API, datastore and using the existing HSM crypto functions.
10. Develop an internet-based closed loop payment system enabling scenarios like the following: A school decides they want to issue cards to the kids in order for the kids to make payments at the school, and possibly at other schools. The school logs onto the system, creates an account, and orders some terminals (because they wish to accept top up payments from open loop cards) and cards. They receive the terminals, deploy them within the school and provide the cards to the kids. The parents of the kids log onto the system, create an account, and link their kids card (issue the card) to their account. They top up their account by making an ecommerce payment or by using a terminal at the school with their open

loop system card/app. Kids can now pay the school and each other. In my view the closed loop cards/apps are simply proxies for initiating value transfer, this value could be associated with actual currencies or virtual currencies like Bitcoin. Payments can be made by:

- a. Card to Phone/Tablet
- b. Phone/Tablet to Card
- c. Phone/Tablet to Phone/Tablet (HCE or QR code)
- d. Terminal to Card
- e. Card to Terminal
- f. Terminal to Phone/Tablet (HCE or QR code)
- g. Phone/Tablet to Terminal (HCE or QR code)

2.2 Acknowledgements

I would like to thank Hough Luff for generously sharing his extensive EMV knowledge with me.

The following open source projects are used by the core libraries:

- Portable.BouncyCastle (<https://www.bouncycastle.org/csharp/>)
- JSON.Net (<https://www.newtonsoft.com/JSON>)

The following open source projects are used by the DC EMV card applet project

- ant-javacard (Ant task for building JavaCard applets) (<https://github.com/martinpaljak/ant-javacard>)
- AppletPlayground (useful libraries to use when building a GlobalPlatform compatible card) (<https://github.com/martinpaljak/AppletPlayground>)
- jcardsim (<https://github.com/licel/jcardsim>)

- Bouncy Castle Provider (<https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk14/1.46>)

The following open source projects are used by the server app

- NSwag (Swagger proxy generation) (<https://github.com/RSuter/NSwag>)
- bc-sharp (<https://github.com/chrishaly/bc-csharp>)
- IdentityServer4 (<https://github.com/IdentityServer/IdentityServer4>)
- Swashbuckle.AspNetCore (<https://github.com/domaindrivendev/Swashbuckle.AspNetCore>)
- Twilio-csharp (<https://github.com/twilio/twilio-csharp>)
- Sendgrid-csharp (<https://github.com/sendgrid/sendgrid-csharp>)

The following web sites provide a lot of useful EMV specifications and/or information.

- eftlab (<https://www.eftlab.co.uk/>)
- emvco (<https://www.emvco.com/>)

I would like to acknowledge the following projects, while they are not used as dependencies in this system they provided insight into how personalization (GlobalPlatformPro) and back end card cryptogram verification work (jPOS). Some routines from these projects were ported from Java to C#.

- GlobalPlatformPro (<https://github.com/martinpaljak/GlobalPlatformPro>)
- jPOS (<https://github.com/jpos/jPOS>)

2.3 Hardware Drivers for Card Readers

DC EMV includes drivers that have been written to interface with NFC hardware on Android, Windows and Windows IOT. The software has been tested on Windows using PC/SC drivers, on Android using both built in NFC on tablets like the Samsung Active, Phones like the Google Nexus and with the ACS1255 Bluetooth NFC card reader. It has been tested on a Raspberry Pi with the OM5577 NFC I2C based card from NXP. This NXP silicon implements the NCI interface and a I2C NCI based driver has been developed to interact with this hardware. All card reader drivers implement the same interface and so the appropriate drivers for the hardware / application platform being used must be configured in the respective application start-up code.

2.4 EMV Kernels and Terminal Libraries

All code, other than specific hardware drivers, and the platform specific application start-up code are written in C# based on .Net Standard. This means the code can be used, unmodified on any of the platforms supported by Xamarin.

The closed loop payment demo app and the personalization application are written using Xamarin Forms. The bulk of the code is in a shared project and therefore is usable across platforms without modification. Xamarin is supported on Windows, Android and iOS. Card drivers have been implemented for Android and Windows UWP and therefore there are Xamarin start-up code projects for Windows UWP and Android only. NFC hardware is not accessible in iOS however the ACS1255 Bluetooth NFC card reader is supported in iOS with an ACS iOS driver which can be easily wrapped as a card driver for this system, in the same way the Android ACS1255 library was.

EMV kernels have been developed implementing kernel contactless specifications 1, 2 and 3 and the contact kernel specification. EMV chose to implement separate contactless kernels specifications to be used by different card associations rather than have a common kernel specification as they have for contact. These kernels have been developed to implement all the mandatory requirements and almost all the optional requirements.

Layered above the EMV kernels are contact and contactless terminal components. These terminal components provide an easy way to embed an EMV payment flow into an application.

EMV QR Code functionality, as per version 1.1 of the EMVCo specification, has been implemented into the terminal components.

2.5 EMV Card Applets

An online only, contactless, EMV kernel 3 compatible card applet has been developed. This is what is meant as the closed loop payment system card mentioned previously. When using this closed loop payment card applet within the closed loop payment system to make a payment, the card returns a signature (cryptogram), which is validated on the server application. The server application uses the EMV security module to manage keys, basically an in memory HSM, and can therefore recreate the card key and validate the card cryptogram. This security module is the same one used by the personalization application to generate the card keys that are injected into the card applet during the personalization process described above.

2.5.1 HCE

An HCE version of the Card Applets for Android has been developed. The user's device can then be used in place of the card in doing payment transaction.

2.6 The Closed Loop Payment Demo System

2.6.1 Demo Client App

A closed loop payment Xamarin app has been developed implementing the following features:

- Transfer of funds from card (account linked to card) to account (account linked to app).
- Transfer of funds from account (account linked to app) to card (account linked to card).
- POS screens for small merchants to manage inventory and sales.
- A Top-Up function which allows EMV cards outside the closed loop payment system to be used to top up the users account
- Account and card administration

These features are implemented by using the Hardware Drivers, EMV Kernels, EMV Terminal Applications and the EMV Card Applet described above.

2.6.2 Demo Server Software

This closed loop payment app calls web services. These web services are implemented as an ASP.Net Core application and have been tested in a Kubernetes cluster running in Azure. The server application uses SQL Server for data persistence. The server application can process transactions as an Acquirer for cards outside the closed loop payment system or as the Issuer for the closed loop payments system cards.

2.6.2.1 External Switch Drivers

When the server application is acquiring transactions from EMV cards issued outside the closed loop payment system, it can forward these transactions on to a SPDH capable switch for transaction authorization. Obviously, this implies some sort of business relationship between the

closed loop system provider and the financial institution who issued the EMV card. The switch could also be that provided by a card association or a country specific interbank switch.

2.7 Tools Used

2.7.1 Visual Studio

Visual Studio 2017 is used for all the projects except the card applet project. All dependencies are installed via Nuget. If you plan on only running with the emulator then you can use a VM, and download an evaluation version of windows preinstalled with Visual Studio at <https://developer.microsoft.com/en-us/windows/downloads/virtual-machines>

2.7.2 SQL Server

MS SQL server is used for the server apps persistence layer. More info on the server app can be found in the

Run Server App section.

2.7.3 IntelliJ

IntelliJ is used for building/running the card applet. More info on the applet can be found in the

EMV Card section. The ant-javacard project is used to build the cap file, essentially the jar file that is loaded onto the JavaCard. The card applet can also be run in an emulator. This is useful when debugging applets. It also means that you can experiment with the system without needing a card reader or a physical card. Figure 2 shows this setup.

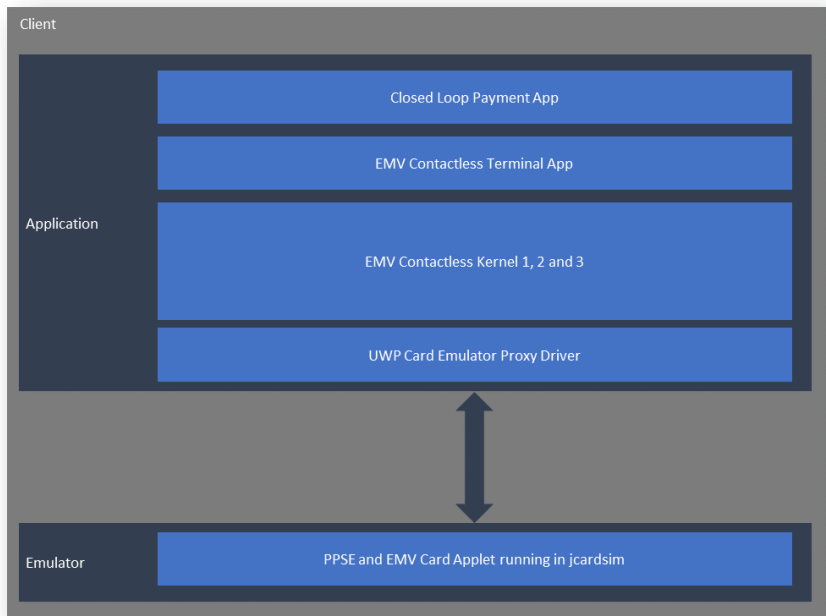


Figure 2

2.7.4 NSwag

NSwag is used to build the proxy class used by the apps to call the JSON based web services on the server app. The proxy class is included in the client application.

3 EMV Core Libraries

The core software has been structured in 3 distinct layers. Messages are passed between the kernel, terminal application and the card interface manager via messaging queues. There is an input q and an output q between the kernel and the terminal application and between the kernel and the card interface manager. The kernel queues and the card interface manager queues are managed by separate threads. A demo closed loop payment app, via the payment UI control, initializes both a contact and contactless terminal application and subscribes to various events offered by the terminal application. The demo app then starts a transaction request on each terminal application. The application that detects the card on its card interface then continues processing the transaction. Figure 3 shows this configuration.

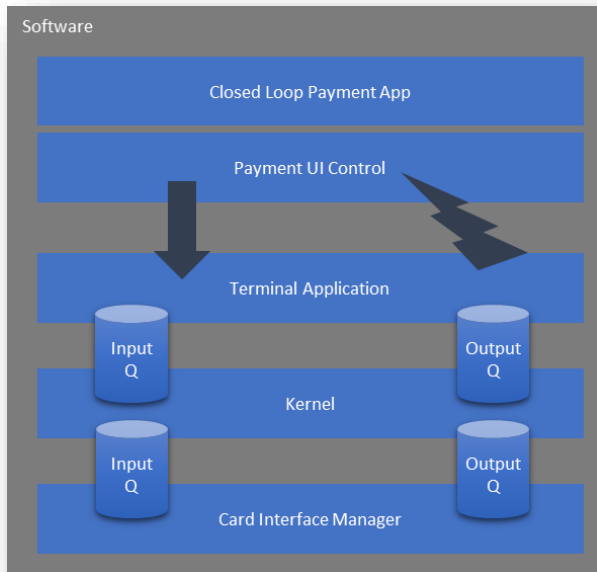


Figure 3

3.1 Hardware Drivers for Card Readers

3.1.1 Windows (PC/SC)

The PC/SC is a spec that card reader manufacturers implement, and therefore software running on Windows becomes interoperable with many different card readers. This works well, and most development was done using a PC/SC card reader. Depending on the reader it is possible that if the terminal takes too long with a procedure (e.g. going online) the card reader will reset the connection to the card and the next ADPU transmitted to the card will fail resulting in the transaction failing.

3.1.2 Android (Built In)

Android devices with built in NFC hardware can be used. The only shortcoming to using this approach is that the card has to be very close to the NFC antennae, often the card needs to be touched to the back of the device, in a specific position in order for the NFC to work.

3.1.3 Android (ACS1255)

A driver for the ACS1255 card reader from ACS has been written. It wraps the low-level Java driver from ACS as a Xamarin library. A fairly complex state engine needed to be written to maintain connection to the device over Bluetooth. Communication to the card over Bluetooth seems a little slower than wired card readers. The range of the NFC of the card reader is far better than that of the built in NFC hardware.

3.1.4 Windows IOT (NXP OM5577)

The driver for the NXP OM5577 card uses I2C to communicate with the card, so, in theory, this driver could be run on any .NET Standard platform on hardware with a suitable I2C driver. It has been tested on a Raspberry Pi 3 with Windows IOT running the closed loop payment app. The PN7120 silicon on the OM5577 board is NCI compliant. The

NCI spec is an NXP spec. The standard has been used by a few other vendors. The spec was probably started with the intention of creating interoperability of devices between different manufacturers of silicon, but over time it hasn't gained much traction and rather than updating the spec with new features, NXP has implemented many propriety extensions, thereby decreasing the chances of interoperability of drivers between different silicon manufacturers.

3.1.5 TPM

A very basic TPM driver was written. This uses the Microsoft TSS library. This driver will work on the Raspberry Pi with a TPM module installed. It can be used to encrypt or sign and decrypt or verify communications to the device, using a key, only known to the TPM. This is really just a demo driver and the provisioning process should interact with an HSM. If asymmetric encryption is used, the then need for a TPM on the EMV terminal is not needed, in my opinion.

3.2 EMV Terminal Applications

There are 2 applications, a contact and contactless application. The application is responsible for transaction pre-processing. The outcome of transaction pre-processing is a selected application on the card. See the EMV pre-processing specifications for more information. Both applications implement transaction pre-processing in accordance with the EMV pre-processing specifications.

3.2.1 EMV Terminal Configuration

The terminal applications are configured via xml. This config can be code based or file based. The code-based driver is configured in the code. The code-based driver is convenient when debugging apps on multiple platforms, however it is not the recommended approach for a production system. If using the code-based approach, the xml is retrieved from the CodeData.cs file in the DCEMV_ConfigurationManager project. For the file-based approach there are both an Android and a Windows UWP driver for loading config from xml files on each platform. If the file driver cannot find the config files it will write a new set of files based on the values in the CodeData.cs file.

3.2.1.1 TerminalConfigurationData.xml

TerminalConfigurationData in CodeData.cs

This file stores global default (contact and contactless) values for terminal configuration. If during the transaction the kernel requires terminal data to continue it will request it of the terminal and the terminal will retrieve it from this list and pass it back to the kernel.

3.2.1.2 TerminalSupportedContactlessRIDs.xml

TerminalSupportedContactlessRIDs in CodeData.cs

For the contactless terminal application, the terminal will use this file to get a list of supported RID's per transaction type. For each transaction type and RID combination there are various flags that can be set which would influence the outcome of the transaction. This is also where a RID is mapped to a specific kernel. The flags that can be set are as follows:

```
<TransactionTypeEnum>PurchaseGoodsAndServices</TransactionTypeEnum>
<StatusCheckSupportFlag xsi:nil="true" />
<ZeroAmountAllowedFlag xsi:nil="true" />
<ReaderContactlessTransactionLimit xsi:nil="true" />
<ReaderContactlessFloorLimit xsi:nil="true" />
<TerminalFloorLimit_9F1B xsi:nil="true" />
<ReaderCVMRequiredLimit xsi:nil="true" />
<ExtendedSelectionSupportFlag xsi:nil="true" />
<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
```

3.2.1.3 TerminalSupportedContactAIDs.xml

KernelGlobalConfigurationData in CodeData.cs

For the contact terminal application, the terminal will use this file to get a list of supported AID's and whether full or partial selection is allowed for this AID.

3.2.2 Contact and Contactless Terminal Application

Each application implements pre-processing as per EMV pre-processing specifications, however beyond this they offer the same interface to an app using either one. The app offers various events that can be subscribed to. The events are:

```
UserInterfaceRequest;
PinRequest;
TRMRequest;
OnlineRequest;
ProcessCompleted;
ExceptionOccured;
```

One key difference is that the contact app will possibly fire the go online request and then the process completed event, however the contactless application will fire the process completed event and then the terminal decides whether to go online or not based on the contactless transaction outcome.

3.3 EMV Kernels

The kernels were implemented according to EMV Co specifications, available online. Where possible the code is annotated with region blocks, where the region description maps to the exact section of the specification, where the requirement is documented.

3.3.1 EMV Kernel Configuration

The kernel is configured via xml and used the same driver as described in EMV Terminal Configuration section.

3.3.1.1 PublicKeys.xml

Certificate Authority Public Key Certificates (Certs in CodeData.cs)

The first of these are the CA public certificates. These will be added as needed, depending on which cards the implementor intends on accepting payments from.

3.3.1.2 RevokedPublicKeys.xml

RevokedCerts in CodeData.cs

Any certificates no longer valid can be added to this file. Any cards requiring a cert loaded in this file will have their ODA or Pin verification fail.

3.3.1.3 ExceptionFile.xml

ExceptionFile in CodeData.cs

The pan of any cards not to be accepted by the system can be stored in this file. Currently only Kernel 3 does the exception check.

3.3.1.4 KernelGlobalConfigurationData.xml

KernelGlobalConfigurationData in CodeData.cs

Global configuration data for the contact kernel. Not all kernels have a global configuration file. Global config are flags that can be set which influence the behaviour of the kernel. Currently this version of the contact kernel has no flags that can be set.

3.3.1.5 KernelConfigurationData.xml

KernelConfigurationData in CodeData.cs

Stored in this file are default kernel values that are added to the contact kernel database at the start of every transaction. Future versions may allow this to be configured per transaction type.

3.3.1.6 Kernel1GlobalConfigurationData.xml

Kernel1GlobalConfigurationData in CodeData.cs

Global configuration data for the contactless 1 kernel. The only flag that can be set for this kernel is the:

```
<VLPTerminalSupportIndicator>true</VLPTerminalSupportIndicator>
```

3.3.1.7 Kernel1ConfigurationData.xml

Kernel1ConfigurationData in CodeData.cs

Stored in this file are default kernel values that are added to the kernel 1 contactless database at the start of every transaction. Future versions may allow this to be configured per transaction type.

3.3.1.8 Kernel2ConfigurationData.xml

Kernel2ConfigurationData in CodeData.cs

There is no global configuration data for the contactless kernel 2.

Stored in this file are default kernel values that are added to the kernel 2 contactless database at the start of every transaction. Future versions may allow this to be configured per transaction type.

3.3.1.9 Kernel3GlobalConfigurationData.xml

Kernel3GlobalConfigurationData in CodeData.cs

Global configuration data for the contactless kernel 3.

```
<IDSSupported>false</IDSSupported>
<FDDAForOnlineSupported>true</FDDAForOnlineSupported>
<SDAForOnlineSupported>true</SDAForOnlineSupported>
<DisplayAvailableSpendingAmount>true</DisplayAvailableSpendingAmount>
<AUCManualCheckSupported>true</AUCManualCheckSupported>
<AUCCashbackCheckSupported>true</AUCCashbackCheckSupported>
<ATMOfflineCheck>true</ATMOfflineCheck>
<ExceptionFileEnabled>true</ExceptionFileEnabled>
```

3.3.1.10 Kernel3ConfigurationData.xml

Kernel3ConfigurationData in CodeData.cs

Stored in this file are default kernel values that are added to the kernel 3 contactless database at the start of every transaction. Future versions may allow this to be configured per transaction type.

3.3.2 EMV Contactless Kernel 1 and 3

Kernel 1 and 3 are implemented as separate kernels however they share a lot of functionality and therefore code.

3.3.3 EMV Contactless Kernel 2

Kernel 2 is a complex kernel and was the initial kernel implemented. This is why a lot of functionality, where possible, was reused from this kernel, including re-use for the contact kernel.

3.3.4 EMV Contact Kernel

This was implemented after the contactless kernels and therefore was written to re-use as much functionality as possible from the contactless work.

3.3.5 EMV QR Codes

An implementation of EMV QR codes has been added. The low-level protocol implementation can be found in the DCEMV_QRDEProtocol project and the higher level EMV QR Code implementation can be found in the DCEMV_EMVProtocol project. The DCEMV_DemoApp and the DCEMV_DemoEMVApp have been updated to include this functionality. A payment can now be made by 2 parties who have the app, one will present their QR code and the other will scan and process the transaction. The presenter of the QR code can then poll for the transaction status. The QR code functionality is multi-platform and will work on all currently supported platforms.

4 EMV Card Applets

There are 2 applets written for loading on a JavaCard to support EMV based contactless transactions.

4.1 PPSE Applet Features

The first is the PPSE application that returns to the terminal a list of applications installed on the card. In this case the applet simply returns a list of 1, the Payment applet described below.

4.2 DCEMV Payment Applet Features

This is an EMV Kernel 3 compatible, online only payment applet. This means an implementor can configure kernel 3 with the RID personalized for the payment applet, via the XML configuration files, and when tapped the card will work with kernel 3. See the

Run Demo App section for an example of the configuration.

In the closed loop payment demo system this payment applet is used, and cards would be issued with this applet installed and personalized. The card generates a cryptogram that is then sent to the demo server application, where it is validated, and if validation passes, the transaction can be completed on the server.

In the

Using the System section there is a tutorial on building and then loading and personalizing a physical card with these applets.

4.3 HCE version of Card Applets

The HCE version of the Card Applets is implemented as an Android HCE Service in the DCEMV_AndroidHCEDriver project. This behaves in the exact same way as the JavaCard version. The perso values are kept in a static class (PersoAndCardStateStorage). The app would need to be extended to allow for creating virtual cards and selecting one as the default. This default card's perso values would then be used during an HCE transaction. The AndroidHostCardEmulator must be selected as the default payment service in the Tap and Pay settings of the device the App is installed on. The device can then be used at any time in place of a card.

5 EMV Card Personalization Application

The DC EMV Card Personalization app was written in order to allow the managing of DC EMV Card applets on a JavaCard, but many of its features can be used to manage any kind of applet on a JavaCard.

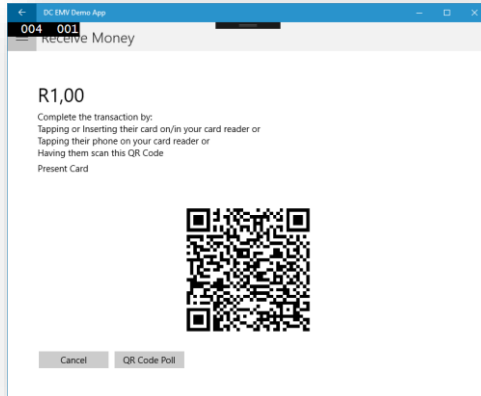
In order to personalize the card, 2 things need to be known, the AID of the security domain applet on the card and the master key of the security domain. Any JavaCard supporting the Global Platform will come pre-configured with a security domain applet and the master key will be available from the manufacturer / seller of the cards.

The features available on the Personalization Application are:

- View Card Data
- View/Remove Applications
- Cap Load
- Install Applet
- Remove/Cap/Install
- Personalization App
- Test App
- Settings

See the In order to present the barcode, install the DCEMV_DemoApp on another device and select the send money option.

1. Enter the amount and press the next button. The QR code will be presented on the next view.



2. Once the scanner of the barcode has completed the transaction, press the QR Code poll button to see if the transaction was completed and if it was approved or declined.

Run Personalization App section for a detailed example.

6 Demo Closed Loop Payment System

The closed loop payment demo system consists of the client app, runnable on Android, Windows or Windows IOT and a server app runnable standalone or in any Docker/Kubernetes environment.

6.1 Demo Client App

The client app allows a user to transfer funds into or out of their account with other users of the closed loop payment system. It offers the following features:

- Online account creation, with email (SendGrid) verification and mobile phone (Twilio) verification.
- Account Management (Forgot Password, Change Password, Update Phone Number)
- Send Money (Sends money from the account associated with the app being used to the account associated with the DCEMV card being tapped on the device)
- Receive Money (Money is sent from the account of the DCEMV card being tapped on the device to the account associated with the app being used)
- Top-Account (An open loop credit or debit card is tapped on the device and the account is topped up with the amount selected)
- Sell Stuff (The user uses a POS style screen to create a shopping basket of items they are selling, and can then take a payment from the DCEMV card, in the same way as Receive Money)
- Manage Inventory (Inventory items can be added, removed, updated and associated with Inventory groups which can be added, removed, updated. Inventory groups are used on the

POS screen as one of the ways for navigating the list of inventory items)

- Card Admin (DCEMV payment cards can be associated with the account associated with the app being used, cards can be disabled or removed from the account)
- Account Transactions (An account statement)

6.1.1 Demo Client Server Proxy Generation with Swagger

If you are modifying the server web services and need to recreate the client proxy file from the demo server swagger definition file, then do the following:

1. Install NSwagStudio using the MSI file on Windows. (v11.17.0.0) (NSwag is used to generate the client proxy from the swagger definition created by the Demo Server app.)
2. Run the server app in standalone mode.
3. Run NSwagStudio, open the file DCEMVDemoServerClient.nswag in the DCEMV_DemoApp folder.
4. Fill in the Specification URL (e.g. <http://{IP/domain}:44359/swagger/v1/swagger.JSON>).
5. Click generate outputs
6. Replace the code in the DCEMVDemoServerClient file in the Proxies folder of DCEMV_DemoApp with the generated code.

6.2 Demo Server Application

6.2.1 Architecture

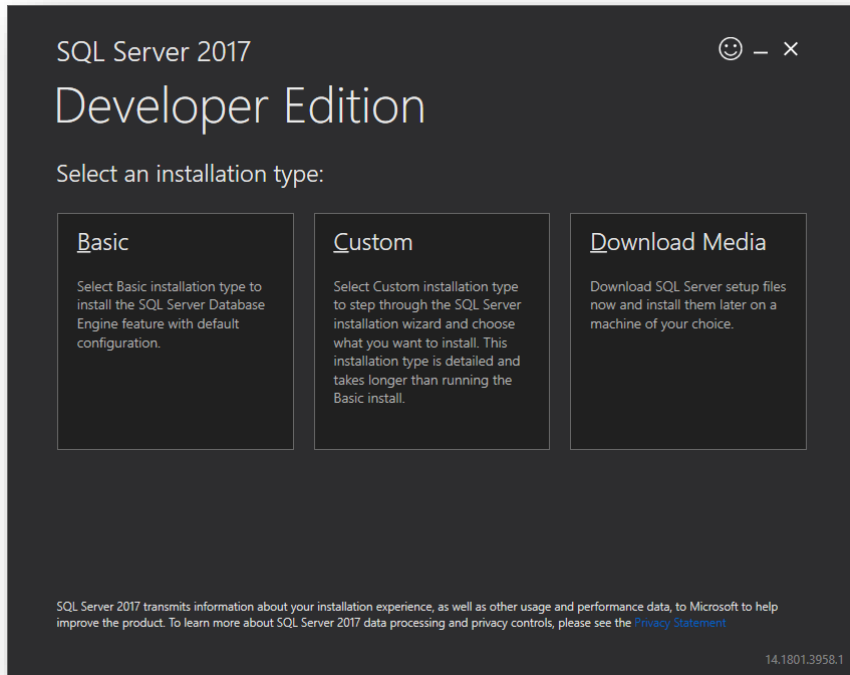
The server is an ASP.NET Core application, integrated with the IdentityServer4 libraries. The client authenticates to the server using a token, which when expires, can be refreshed. The token is retrieved by the client app once using the username (email address) and password of the user.

The server implements JSON based web services and using the Swashbuckle library creates a Swagger definition file as well as UI for testing the various web services. The web services implement the business functionality that the client app makes available to a user. If running in Docker or Kubernetes the server app is deployed behind a reverse proxy which terminates SSL. NGINX is used for the reverse proxy. The server application and the reverse proxy are deployed within Docker images. These images can be deployed in a Docker or Kubernetes environment.

The server uses a MS SQL database to store state for the IdentityServer4 implementation, which in turn is integrated with ASPIdentity.

6.2.2 Installing the Database

1. Download SQL Server 2017 Developer Edition.
2. Install it using the basic installation.



3. Install SQL Server Management Services once the SQL Server installation completes by clicking Install SSMS. This will redirect you to a MS Web page where you can download the installation. Install SSMS by running the file downloaded.

SQL Server 2017

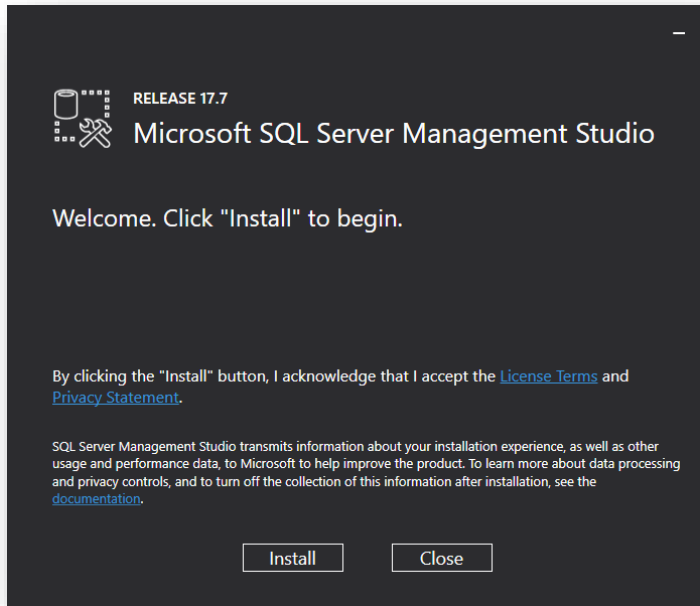


Developer Edition

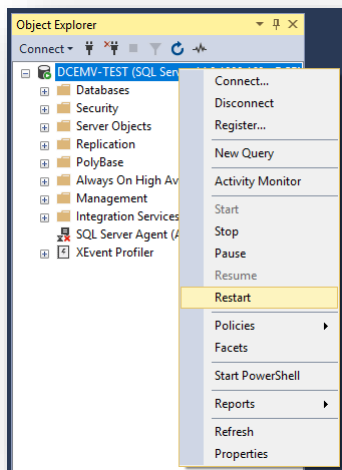
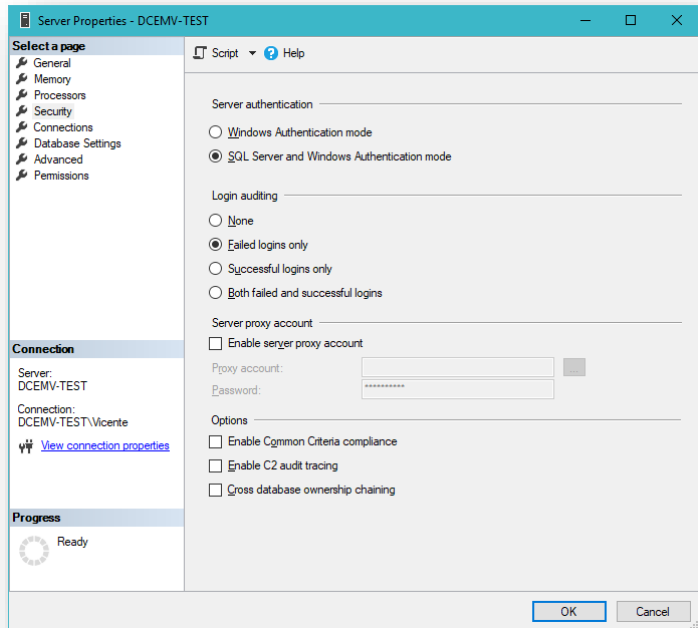
Installation has completed successfully!

INSTANCE NAME	CONNECTION STRING
MSSQLSERVER	Server=localhost;Database=master;Trusted_Connection=True;
SQL ADMINISTRATORS	SQL SERVER INSTALL LOG FOLDER
DCEMV-TEST\Vicente	C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log\2018051
FEATURES INSTALLED	INSTALLATION MEDIA FOLDER
SQLENGINE	C:\SQLServer2017Media\Developer_ENU
VERSION	INSTALLATION RESOURCES FOLDER
14.0.1000.169, RTM	C:\Program Files\Microsoft SQL Server\140\SSER\Resources

- [Connect Now](#)
- [Customize](#)
- [Install SSMS](#)
- [Close](#)



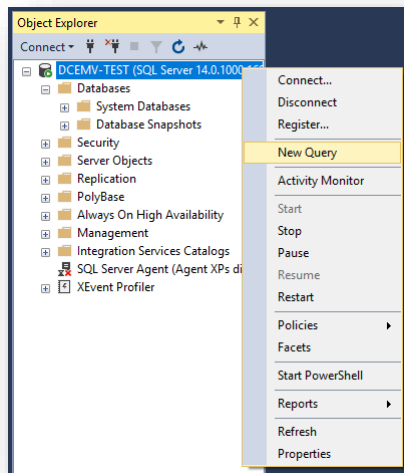
4. Enable mixed mode authentication in SQL Server using SSMS.
Restart SQL Server.



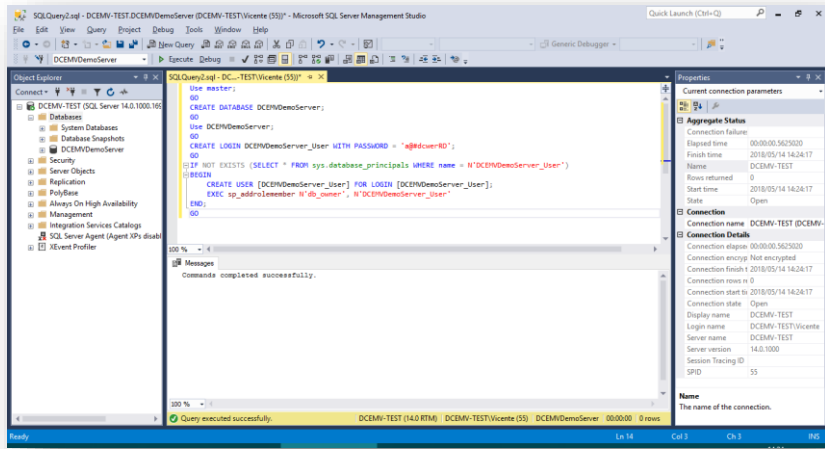
6.2.3 Creating the Database

The first task in configuring the database is creating an empty database as well as the user accounts that will be used by the server app to access the database. Use the create_DB_and_users.sql file to do this.

1. Open SSMS, login with your local account credentials (the default if basic installation used). Create a new query window.



2. Paste the script into the new query window and click Execute.



3. The empty database and users have been created.

6.2.4 Configuring the Database

Entity Framework is used for persisting changes to the database. In order to run the Entity Framework scripts, the DotNetCliToolReference xml config must exist in the project file of the project file containing the Entity Framework Migrations, in this case the DCEMV_DemoServer project. This entry has already been added to the project file. For reference, an example of the config that needs to be added is in the db_emf_admin_scripts.bat file in the ScriptsDBDeploy folder of the DCEMV_DemoServer project.

Once an empty database and users are created the Entity Framework will take care of creating the database tables. These scripts are in the update_db.bat file of the DeployScripts folder of the DCEMV_DemoServer project.

1. Set DB_SERVER_NAME to your IP/domain name/localhost in *update_db.bat*. if the database is installed on your local machine then the default setting of localhost will be fine.
2. Open a command prompt in the DCEMV_DemoServer folder.
3. Run the batch file update_db.bat (e.g. type in *call ScriptsDBDeploy/update_db.bat* from the DCEMV_DemoServer folder in the command prompt.)
4. If successful, your database will contain empty database tables for the DC EMV Closed Loop Payment System.
5. The Connection string used by the DC EMV Demo Client application is in the appsettings.json file. This will not need to change if you used the scripts specified above with their default values.

6.2.5 Running the Demo Server App

The demo app server can be run:

- standalone
- as a Docker image using Docker for Windows
- as a Docker image deployed in a Kubernetes cluster in Azure.

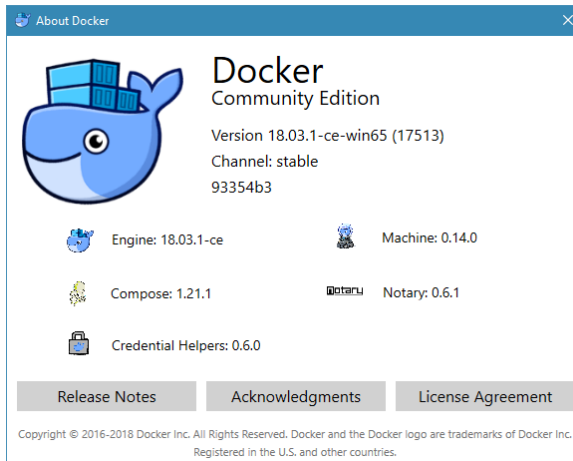
6.2.5.1 Standalone

See the

Run Server App section for a tutorial on how to run the server application in standalone mode.

6.2.5.2 Docker For Windows

The server app can be run in Docker. It has been tested on Windows running Docker for Windows using Linux based containers.

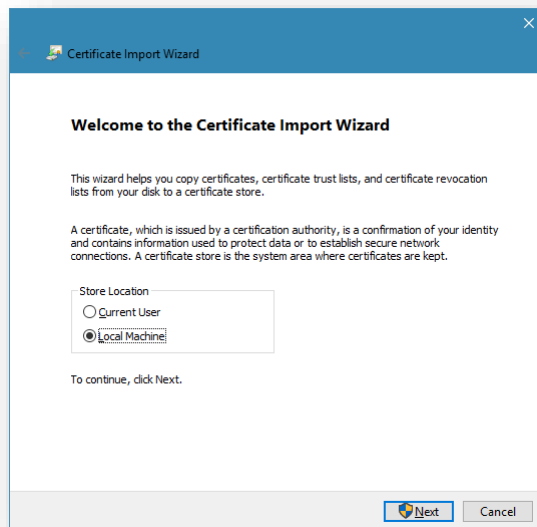


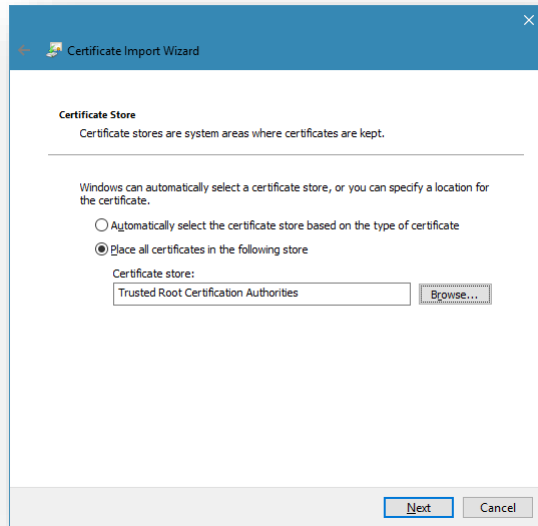
The server app is deployed behind a NGINX reverse proxy which terminates SSL. The ScriptsReverseProxy folder in the DCEMV_DemoServer project contains the Dockerfile used to build the reverseproxy Docker image. The Dockerfile in the DCEMV_DemoServer project root folder is used to build the dcmvdeoserver Docker image. Both images are deployed to Docker / run when running the Docker compose project via the docker-compose.yml file.

1. Install Docker for Windows
2. Make sure Docker is re-started if your IP address changes
3. Make sure you are connected to the internet as it may need to download base images
4. Create a certificate tied to your IP address/domain, use the *makeTLSCerts.ps1* PowerShell script in the

ScriptsReverseProxy folder. Before running it, update it with your IP address/domain details. Include the alt name in the cert as this is what is used on the Android platform to validate the cert name.

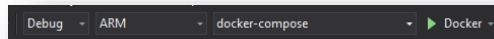
5. Trust the cert on the machine/device you will run the client app from, as this is a self-signed certificate. Place the cert in the local machine, trusted root authority store on windows
 - a. On Windows install the cert by right clicking on it and selecting install certificate



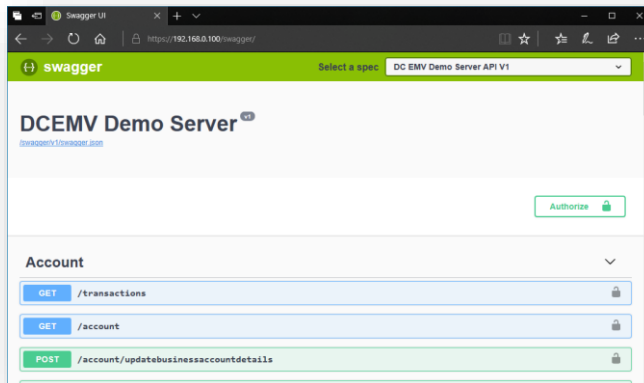


- b. On Raspberry Pi, SSH to the pi and run the following command `C:\Windows\System32\certmgr.exe -add {certname}.crt -s -r localMachine root -all` and `certmgr -v -s root`
- c. For the Demo Server app to be able to call into the oAuth2 API via the ReverseProxy Docker Image on Linux, the cert must be trusted by the Docker Demo Server image, this is done via the DCEMV_DemoServer Dockerfile file.
6. Update the nginx.conf in the DCEMV_ReverseProxy folder with the cert and key name
7. Update the Docker file in the DCEMV_ReverseProxy folder with the cert and key name
8. Update the Docker file in the DCEMV_DemoServer project with the cert name

9. Update the docker-compose.yml file in the DCEMV_DemoServer project with the IP address/domain name for both the id server and the database, the server name here must match the server name used when generating the certificate.
10. Set the docker-compose project as the start-up and run it in Visual Studio



11. Check that the server is running correctly by opening the following URL in your browser, replacing IP/domain with your details. <https://{IP/domain}/swagger>



The client app can now be configured with the server name. This is described in the

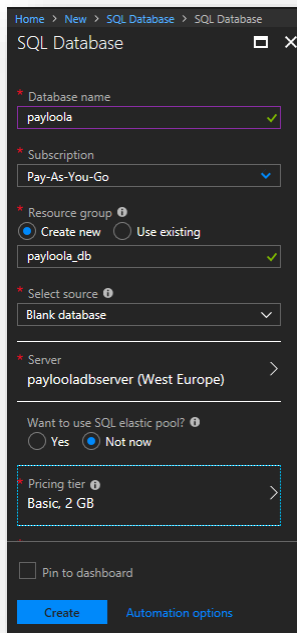
Run Demo App.

6.2.5.3 Azure

This example below uses the domain www.payloola.com domain as an example, you will have to use your own.

6.2.5.3.1 Installing the Database

Before deploying the app in Azure, first create a MS SQL database in Azure.



Home > New > SQL Database > SQL Database

SQL Database

* Database name
payloola ✓

* Subscription
Pay-As-You-Go

* Resource group ⓘ
☒ Create new ☐ Use existing
payloola_db ✓

* Select source ⓘ
Blank database

* Server
paylolaadbserver (West Europe) >

Want to use SQL elastic pool? ⓘ
☐ Yes ☒ Not now

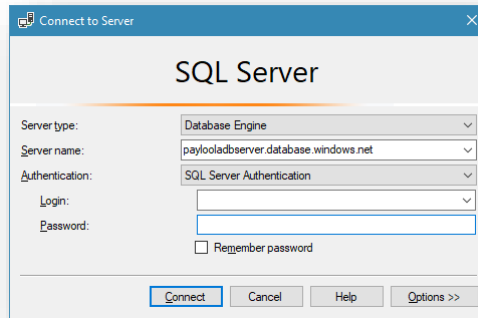
* Pricing tier ⓘ
Basic, 2 GB >

☐ Pin to dashboard

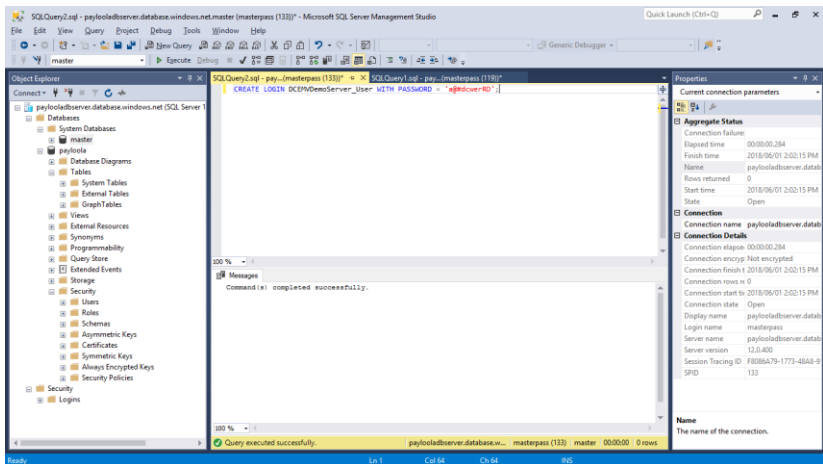
Create Automation options

6.2.5.3.2 Creating the Database

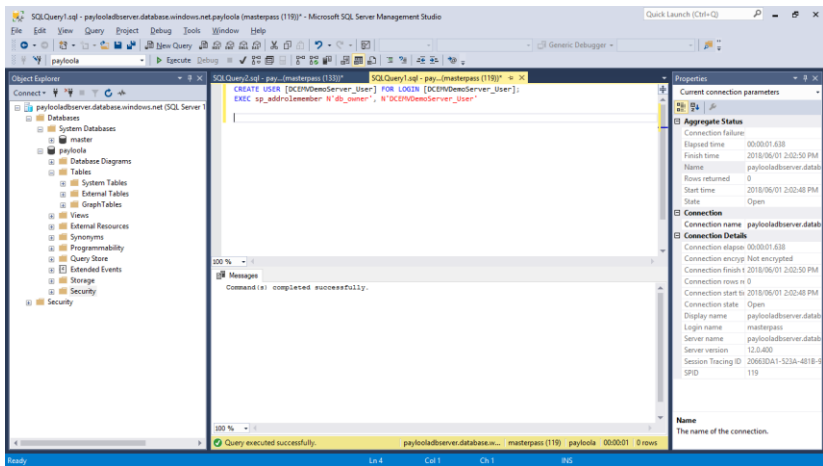
1. Open SSMS, login with your Azure DB account credentials and the Azure DB server connection details. These can be found in the Azure portal, for the DB server resource.



2. Follow the instructions for allowing access to the DB from your IP.
3. Create a new query window on the master DB and another on the payloada DB.
4. On master run the following, this can be copied from the create_db_and_users.sql file.



5. On payloada run the following, this can be copied from the create_db_and_users.sql file.



6. An empty database has been created with the necessary user

6.2.5.3.3 Configuring the Database

1. In the update_db.bat, set the DB_SERVER_NAME value to your Azure DB server instance name, in this example it is paylooladbserver.database.windows.net
2. In the appsettings.json replace the database value with the name of the database, in this example it is payloola
3. Follow the steps in the Configuring the Database section.
4. Replace DB_SERVER_NAME in launchSettings.json with your Azure DB server instance name, in this example it is paylooladbserver.database.windows.net
5. Run the DCEMV_DemoServer in standalone in order for it to update Identity Server settings in the Azure DB
6. Once this is complete the database is ready.

6.2.5.3.4 Create the Azure Kubernetes cluster and the Azure Container Registry

The DCEMV_DemoServer Docker image has been tested in a Kubernetes cluster running on Azure. The ScriptsKubernetesDeploy folder in the DCEMV_DemoServer project contains the scripts needs

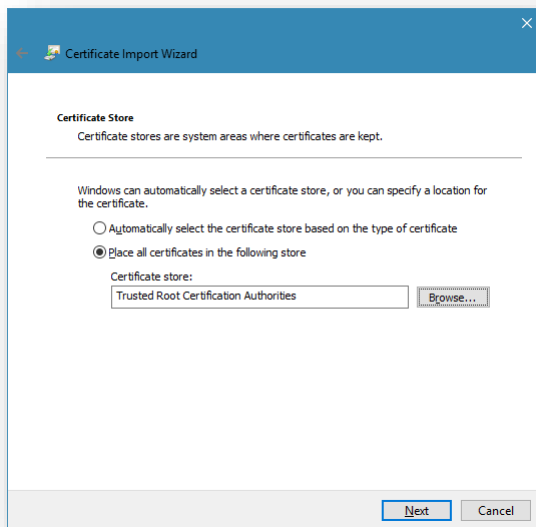
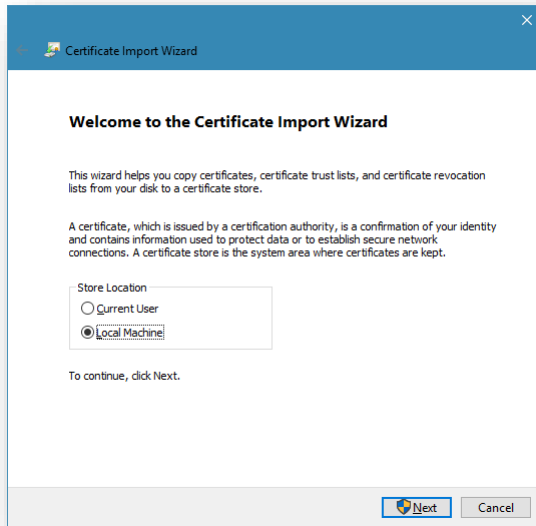
for creating the Kubernetes cluster and the Azure Container Registry as well as for deploying the Docker images to the Kubernetes cluster. The creation scripts are based on the tutorial at <https://docs.microsoft.com/en-us/Azure/aks/>

The versions used are:

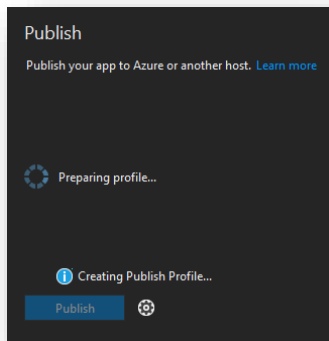
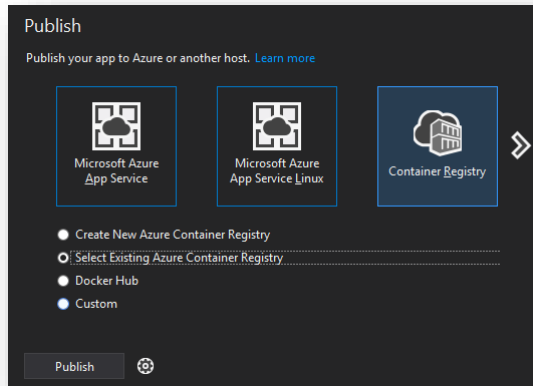
```
Client Version: version.Info{Major:"1", Minor:"9", GitVersion:"v1.9.1",
GitCommit:"3a1c9449a956b6026f075fa3134ff92f7d55f812", GitTreeState:"clean",
BuildDate:"2018-01-04T11:52:23Z", GoVersion:"go1.9.2", Compiler:"gc",
Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"9", GitVersion:"v1.9.6",
GitCommit:"9f8ebd171479bec0ada837d7ee641dec2f8c6dd1", GitTreeState:"clean",
BuildDate:"2018-03-21T15:13:31Z", GoVersion:"go1.9.3", Compiler:"gc",
Platform:"linux/amd64"}
```

6.2.5.3.5 Build the Docker images

1. Create a certificate tied to your IP address/domain, use the *makeTLSCerts.ps1* PowerShell script in the *ScriptsReverseProxy* folder. Before running it, update it with your IP address/domain details. Include the alt name in the cert as this is what is used on the Android platform to validate the cert name. In this example the domain name is *www.payloola.com*
2. Trust the cert on the machine/device you will run the client app from, as this is a self-signed certificate. Place the cert in the local machine, trusted root authority store on windows
 - a. On Windows install the cert by right clicking on it and selecting install certificate



- b. On Raspberry Pi, SSH to the pi and run the following command `C:\Windows\System32\certmgr.exe -add {certname}.crt -s -r localMachine root -all` and `certmgr -v -s root`
 - c. For the Demo Server app to be able to call into the OAuth2 API via the ReverseProxy Docker Image on Linux, the cert must be trusted by the Docker Demo Server image, this is done via the DCEMV_DemoServer Dockerfile file.
3. Update the Docker file in the DCEMV_DemoServer project with the cert name
4. Update the docker-compose.yml file in the DCEMV_DemoServer project with the IP address/domain name for both the id server and the database, the server name here must match the server name used when generating the certificate. In this example the values are <https://www.payloola.com> and `paylooladbserver.database.windows.net`
5. Right click on the DCEMV_DemoServer project in Visual Studio and select publish. Fill in your ACR repository details. Click Publish. This will build the docker image in release mode and push the built image to your Azure Container Registry.



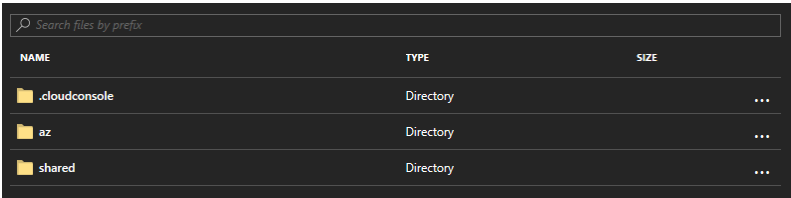
6. Make a note of the tag allocated to the image when it is published.
7. Open the `deployment_dcemvdemoserver-deployment.yaml` and update the following line with the tag value -> image: `payloolacontreg.azurecr.io/dcemvdemoserver:{tag here}`

6.2.5.3.6 Deploy to the Azure Kubernetes cluster

The `ScriptsKubernetesDeploy` folder in the `DCEMV_DemoServer` project contains the scripts needed for tagging and then pushing the

images from your local Docker instance to an Azure container registry. It also contains the script needed for configuring Kubernetes to run the Docker images. The yaml files needed by the commands need to be uploaded to your cloud shell storage. Go to your cloud storage account, click connect, and a view will be displayed showing the Powershell commands needed to be run to map a Z drive on your machine to the share.

Copy the yaml files to the share, the share should contain the folders shown each containing the yaml files.

A screenshot of a cloud storage interface. At the top is a search bar with a magnifying glass icon and the text "Search files by prefix". Below the search bar is a table with three columns: "NAME", "TYPE", and "SIZE". The table contains three rows, each representing a folder. The first row is ".cloudconsole", the second is "az", and the third is "shared". All three are listed as "Directory" type. To the right of each row is a three-dot menu icon.

NAME	TYPE	SIZE
.cloudconsole	Directory	...
az	Directory	...
shared	Directory	...

Execute the commands in the Azure cloud shell from the Configuring the cluster section of the configure-az-k8s.sh file.

6.3 Online Approver Drivers

Currently there are 4 online approver implementations. When a card requests online approval these drivers are used. These drivers are very simple and should not be used as is in a production system, they merely exist to support the demo applications or test the EMV kernels.

The table below summarizes what cards and interface types must be used with the demo client app views.

View	Card Type	Interface Type
Card Admin	DC EMV	Contactless (Emulator or Physical)
Transact	DC EMV	Contactless (Emulator or Physical)
POS View	DC EMV	Contactless (Emulator or Physical)
TopUp	DC EMV or EMV	EMV Contact (Physical) or EMV Contactless (Physical) or DC EMV Contactless (Emulator)

6.3.1 Auth Driver configuration

Auth drivers must be configured on the client and the server. The following tables summarize the possible configurations. The Auth driver configured on the client is used only for contact cards requiring online authorization.

6.3.1.1 Client side

Auth driver configured on client side via MainPage or MainActivity

Top Up done with Contact card which requests Online Auth	SPDHApprover configured	Goes direct to SPDH host for Auth
Top Up done with Contact card which requests Online Auth	DCEMVServerOnlineApprover configured	Calls AuthTransactionToIssuer API on DC EMV Demo Server, and auth depends on what Auth driver is configured on the server
Top Up done with Contact card which requests Online Auth	SimulatedApprover configured	Calculates Authorization locally
Top Up done with Contactless card which requests Online Auth	N/A	Calls TopUp API on DC EMV Demo Server, and auth depends on what Auth driver is configured on the server

6.3.1.2 Server Side

Auth driver configured on server side via GoOnline method in Transaction Controller

Server Side TopUp API called by a Contactless card which requested online auth	N/A	GoOnline Not enabled in code, will approve without going online, this is simpler for demo purposes
Server Side TopUp API called by a contact card which requests online auth (Invalid as the AuthTransactionToIssuer API would have already been called for a contact card)	N/A	N/A
Server Side AuthTransactionToIssuer API called by a contact card requesting online auth	SimulatedApprover configured	Calculates Authorization locally
Server Side AuthTransactionToIssuer API called by a contact card requesting online auth	SPDHApprover configured	Goes to SPDH host for Auth
Server Side AuthTransactionToIssuer API called by a contactless card requesting online auth (Invalid as AuthTransactionToIssuer only called for contact cards)	N/A	N/A

6.3.2 ContactlessDummyOnlineApprover

This driver is used for registering a card to an account (Card Admin View), where only the card number is needed and there is no need to go online.

6.3.3 SPDH Driver

A basic SPDH driver has been implemented in DCEMV_SPDHProtocol. This can be used to test the contact kernel, in terms of getting an approval on 2nd gen ac. It can also be used for getting approvals for contactless transactions requiring online approval.

6.3.4 DCEMVServerOnlineApprover

This driver simply calls the DC EMV Demo server AuthTransactionToIssuer API which in turn authorizes using the auth driver configured on the server.

6.3.5 SimulatedApprover

If using the SimulatedApprover, you need to know the master AC key for the cards you plan on testing with. If you have the keys then this module can be used to generate an ARPC which is passed back to the card during a contact transaction. The upside of using this auth driver is that there is then no dependence on any external service to authorize the transactions during testing. The keys can be configured in the SimulatedApprover class.

6.3.6 ISO8583 Driver

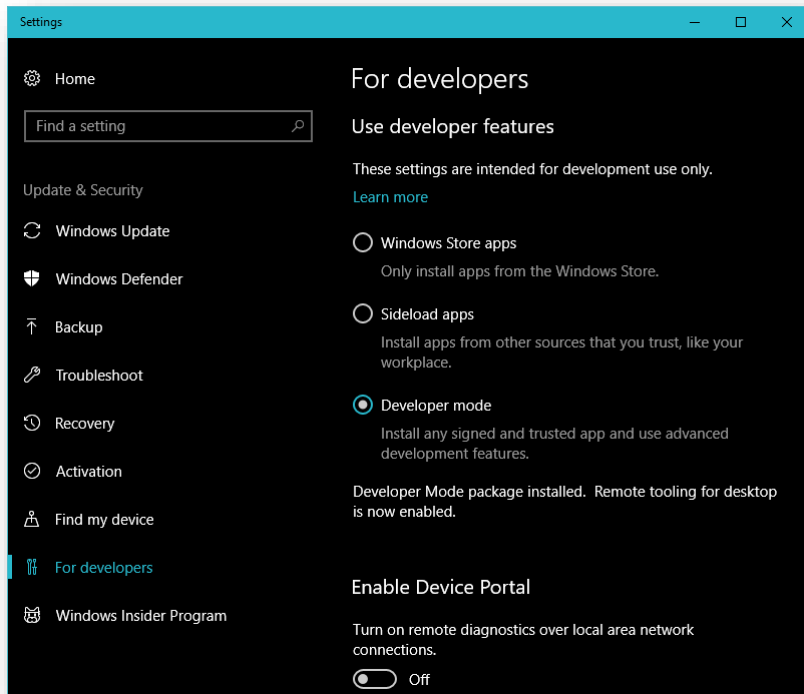
A transaction switch will be implemented in a future release, the SPDH driver fleshed out and an ISO8583 driver will be added.

7 Using the System

7.1 Install Tools

7.1.1 Configure Windows

Ensure developer mode is set in your Windows settings. Windows 10 version 1803 was used for this tutorial.



7.1.2 Docker for Windows

If you plan on running the server app via Docker then docker for Windows needs to be installed. See the following.

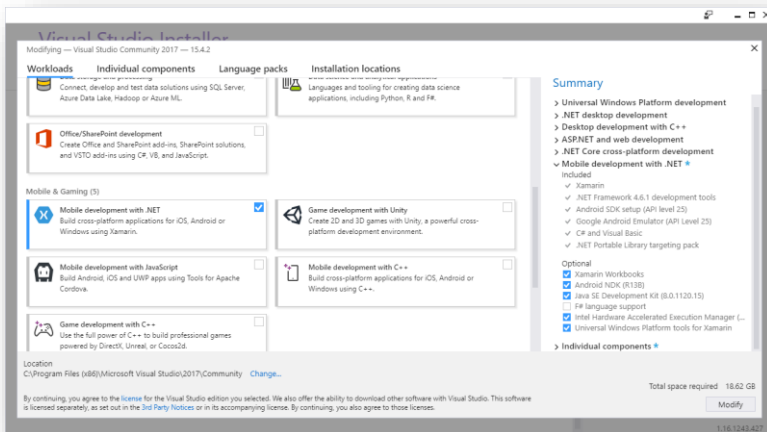
<https://docs.docker.com/docker-for-windows/>

<https://docs.docker.com/docker-for-windows/install/>

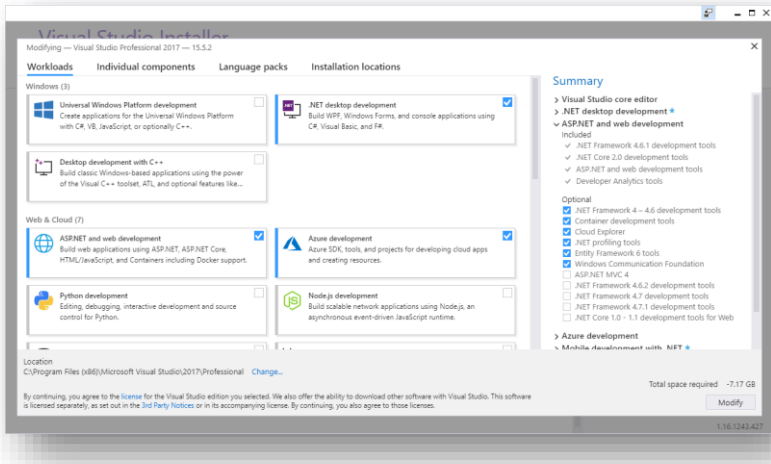
7.1.3 Visual Studio

Visual Studio Community Edition can be used for all projects. Visual Studio Community Version 15.7.5 was used with this tutorial. The windows versions of the various apps are used to demo the software.

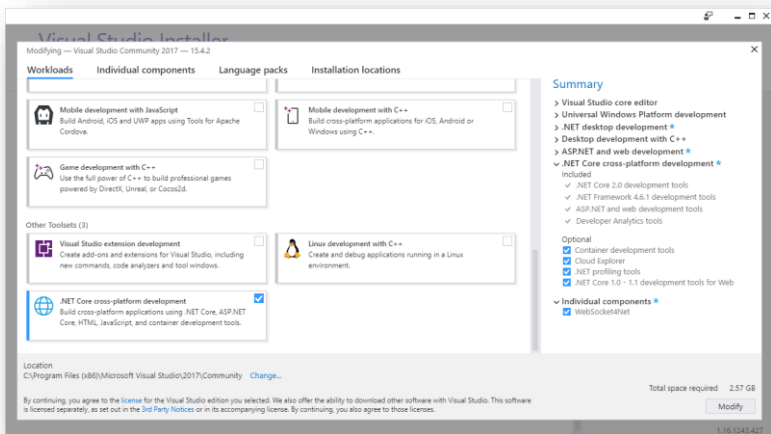
1. Install Visual Studio. Once installed ensure additional components are installed as described below. Open the Visual Studio Installer.
2. In order to run the Android version of the applications the Xamarin components are needed. Open the Visual Studio installer and select the Mobile Development with .NET component.

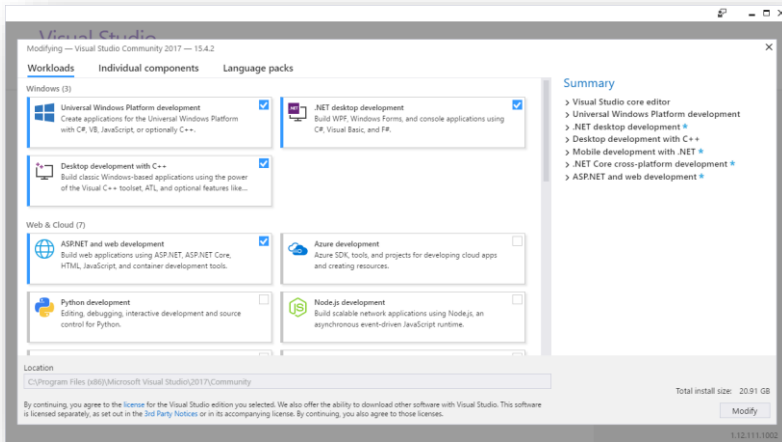


3. In order to run the Docker project, the ASP.NET and web development component is needed. In the Visual Studio installer select the ASP.NET and web development component.

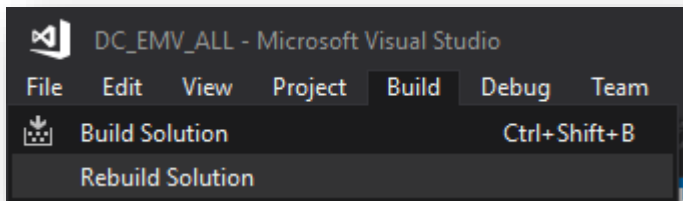


- In order to run the server project, the .NET Core cross-platform development components are needed. In the Visual Studio installer select the .NET Core cross-platform development component. **Also select .Net Core 1.0-1.1 option.**

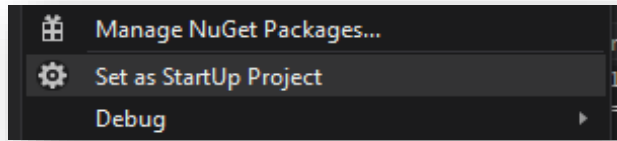




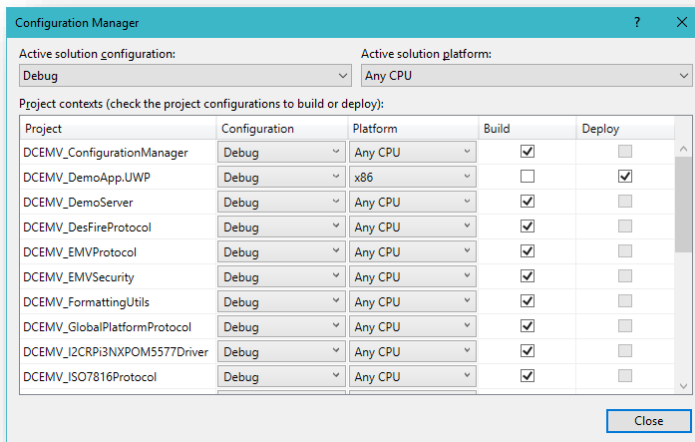
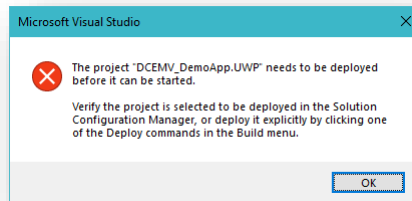
5. Open the DC_EMV_ALL solution. This loads all projects. Other solutions files are included for convenience. When running the full system multiple instances of Visual Studio will need to be run and these solutions only load the projects applicable to the application being run.
6. After opening the solution, the first time, do a Rebuild. You must be connected to the Internet in order for Nuget packages to be downloaded the first time.

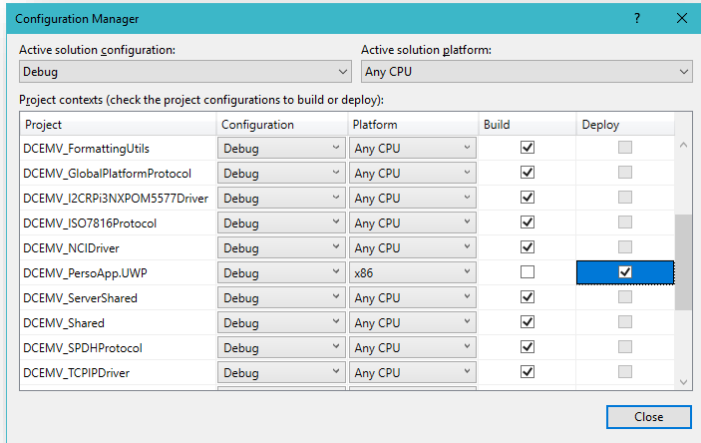


7. Once built, set the project you wish to “set as startup” by right clicking on the project name.

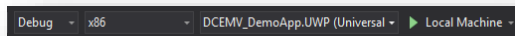


8. If you get the message below, ensure both the build and deploy check box in the configuration manager for both the Demo App and the Perso App are checked.

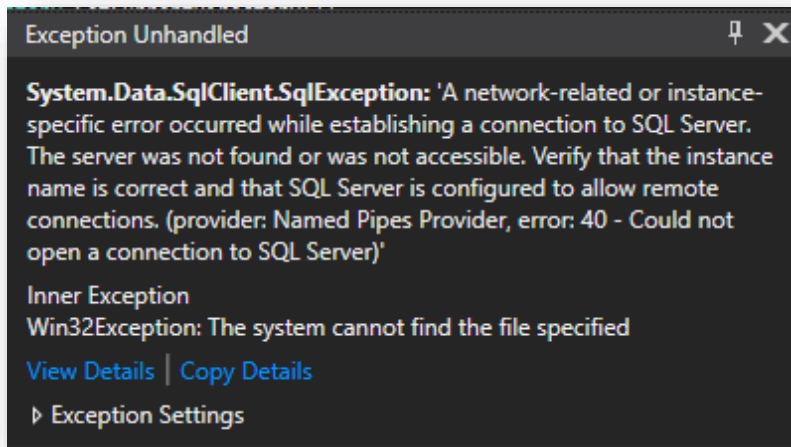




9. Ensure solution platform is set to x86 for the UWP Demo Client and Perso app when trying to run them.



10. Ensure you have installed and configured the database as described in If you haven't you'll get the following error when running the server.

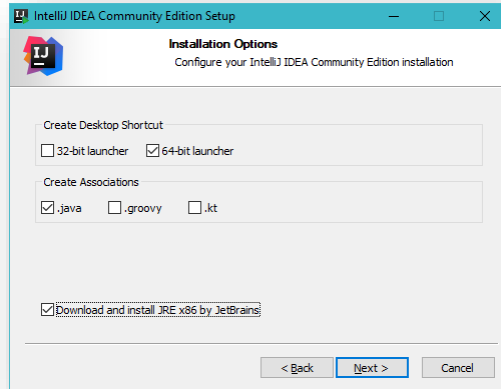


11. Your Visual Studio environment is now configured.

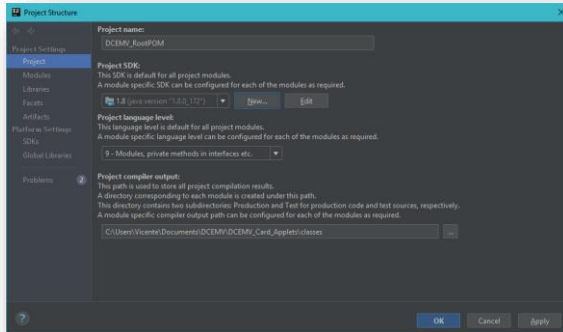
7.1.4 IntelliJ

A Java development environment is needed to build the card applets and to run and debug them in the simulator. The tool used for this is IntelliJ Community Version 2018.1.3. although more recent versions should be fine.

1. Install the JRE x86 if you don't have a JRE/JDK already installed on your machine.



2. Install the JavaCard JDK (Java Card Classic Development Kit 3.0.5u2) from <http://www.oracle.com/technetwork/java/embedded/javacard/downloads/javacard-sdk-2043229.html> .
3. Install a Java JDK from www.oracle.com/technetwork/java/javase/downloads/index.html (Java SE Development Kit 8u172 was used)
4. Once all installed, Open the DCEMV_CardApplets project. If you get prompted to allow access through the firewall, click allow access. Let IntelliJ download any plugins it needs.
5. Set the Project JDK to the Java JDK just installed.



6. Let the indexing process complete.
7. Your IntelliJ environment is now configured.

7.2 Implementing the Card Applets

The card applets can be built in 2 ways, for loading in a simulator or for loading on a physical card. The Java Card Development Kit 3.0.5u1b is used for these applets.

7.2.1 In a Simulator

The applets are run in the jcardsim simulator. The UWP proxy card driver DCEMV_WindowsCardEmulatorProxyDriver when configured in the client relays ADPU requests to the DCEMV_CardReaderEmulator (in DCEMV_Card_Applets) application, via TCP/IP, which in turns sends the ADPU command to the card applets running in the jcardsim simulator.

7.2.1.1 Build Jars

Maven is used as the build system, and the pom files configure IntelliJ.

All dependencies, except jcardsim, are included in the projects, as libraries. The jcardsim project is included as a module as some modifications to its pom were needed to make it work in this project.

1. Open the root pom and set the paths of the globalplatformPathToJar, jcardSimPathToJar and javaCardPathToJDK. The target folder of jcardsim will be created when the jcardsim module is built.
2. Open the jcardsim pom and update the path to the JavaCard JDK of the maven-install-plugin.

Tip: Enable Maven Auto Import on changes.

3. To build jcardsim the first time do the following:
4. Comment out the following in the jcardsim pom

```
<dependency>  
    <groupId>oracle.javacard</groupId>  
    <artifactId>API_classic</artifactId>
```

```
        <version>${jcApiVersion}</version>
        <scope>compile</scope>
</dependency>
```

5. Do a Maven clean on the jcardsim module
6. Do an install on the jcardsim module. This will fail but will copy the jar needed by the dependency to your .m2 repository in .m2\repository\oracle\javacard\API_classic\3.0.5
7. Uncomment the dependency
8. Do a Maven clean on the jcardsim module
9. Do an install on the jcardsim module. This should now compile.
10. In order to configure the card to work correctly in the emulator:
 - the doTestPerso and calcTrack2 methods must be uncommented in EMVCard.java
 - the methods needed by doTestPerso and calcTrack2 in Util.java must be uncommented
 - The call to doTestPerso must be uncommented
11. Now do a maven install on the DCEMV_RootPOM
12. All projects should compile successfully

7.2.1.2 Configure the Emulated Applet

In the EMV Card Applet is a method called doTestPerso. This method personalizes the card with default values.

doTestPerso is configured with:

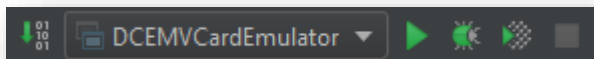
- pan 1234567890123456
- psn 01
- expiry of 2501
- service code 201
- key 8CB9F7D54362B0A240D0AE626780A86B

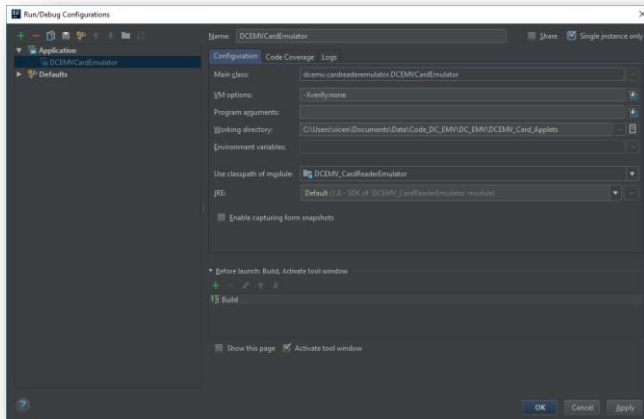
If you wish to change the keys or data values note the following:

- EMVSec_TestGenerateDESKeys generates a master encrypted ac key, and a clear card key derived from the master key, the pan and the pan sequence number
- The tm_ICCACMasterKey personalized in doTestPerso must match the clear key generated for the card. The key selected depends on whether the simulated payment provider, using standard well known test keys, is to be used with DCEMV_DemoEMVApp or if the DC EMV Demo server, using keys generated by the HSM, is to be used.
- The pan and pan sequence number personalized in doTestPerso must match the values used when generating the card key
- The master encrypted ac key is kept on the server (mkACEncrypted in ControllerBase) and passed to the verify cryptogram method.

7.2.1.3 Run Emulator

The main class for running the simulator is dcmv.cardreaderemulator.DCEMVCardEmulator. The -Xverify:none VM argument must be used when running the simulator. Right clicking on the main class and selecting Debug will fail but create the Debug option on the toolbar, which can then be edited as below:





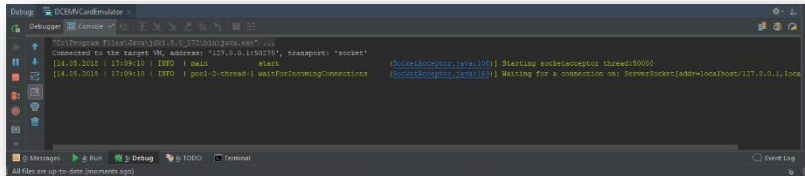
The simulator is started on port TCP/IP 50000 by default. If you change this, make sure Win10CardProxy is configured to send data to the port you change it to. (in the MainPage method of the UWP version or MainActivity of the Android version)

The AID's are configured in startEmulator of DCEMVCARDEmulator. The AID selected here must have an equivalent entry in the kernel configuration. See

Run Demo App.

Remember to do a rebuild (maven clean install for jar(emulator), ant build for cap files) if switching from CAP file building to emulation.

Once running, the Emulator will be waiting for a connection:



7.2.2 On a Physical Card

The card used to create this tutorial supports GP 2.1.1 / VGP 2.1.1 and JavaCard 2.2.1.

7.2.2.1 Build CAP Files

The ant-javacard project is used to simplify the process of building the cap file. The packaging AID's are configured in the MANIFEST.MF and build.xml file. The applet AID's are configured during applet installation after cap file loading using the personalization application. In the personalization application the AID used when installing the applet must be the same as those configured in build.xml.

In order to build the cap files for loading on a card:

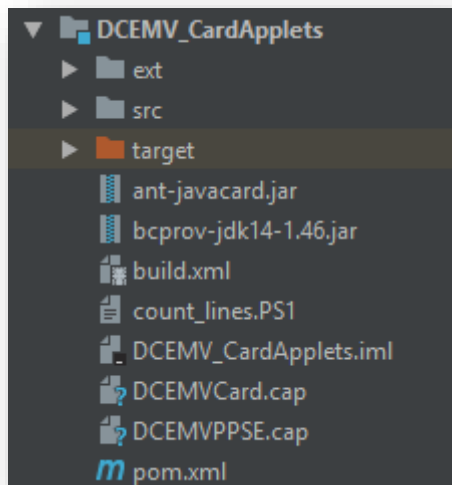
- the doTestPerso and calcTrack2 methods in EMVCard.java must be commented out
- the methods needed by doTestPerso and calcTrack2 in Util.java must be commented out
- The call to doTestPerso must be commented out

To build the CAP files, do the following:

1. Right click on build.xml in DCEMV_CardApplets and select Add as Ant Build File
2. In the build.xml update the JC305 variable to point to your JavaCard JDK
3. Download the Java Card Kit 2.2.1 from:

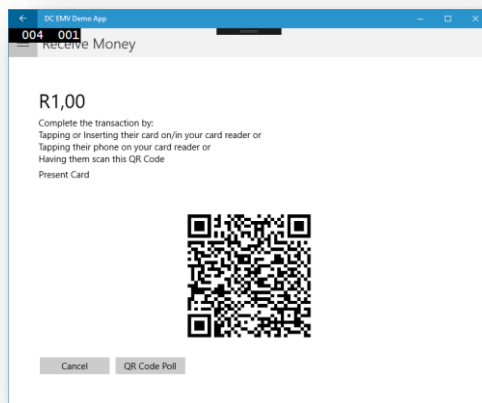
www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javame-419430.html#java_card_kit-2.2.1-oth-JPR

4. Extract it to e.g. C:/Program Files (x86)/Oracle/
5. Update the JC221 variable in build.xml to point to the java card kit 2.2.1
6. Double click the build option in the Ant Tool Window
7. If successful 2 cap files will be created in the root folder of the DCEMV_CardApplets folder.



7.2.2.2 Configure Physical Card

3. Loading of the applet is done using the personalization application. See In order to present the barcode, install the DCEMV_DemoApp on another device and select the send money option.
4. Enter the amount and press the next button. The QR code will be presented on the next view.



5. Once the scanner of the barcode has completed the transaction, press the QR Code poll button to see if the transaction was completed and if it was approved or declined.

Run Personalization App for a detailed example.

7.2.3 HCE on device

The perso values for the HCE version of the card applet are configured in the PersoAndCardStateStorage class of the DCEMV_AndroidHCEDriver project. The key selected depends on whether the simulated payment provider, using standard well known test keys, is to be used with DCEMV_DemoEMVApp or if the DC EMV Demo server, using keys generated by the HSM, is to be used.

7.3 Run Server App Standalone

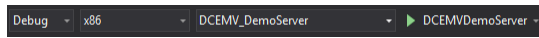
To Enable SendGrid functionality, add the build compilation symbol `EnableSendGrid` to the `DCEMV_DemoServer` project, add your SendGrid emailKey to the `AuthMessageSender` class

To Enable Twilio functionality, add the build compilation symbol `EnableTwilio` to the `DCEMV_DemoServer` project, add your Twilio smsKey, smsAuthToken and smsFromNumber to the `AuthMessageSender` class.

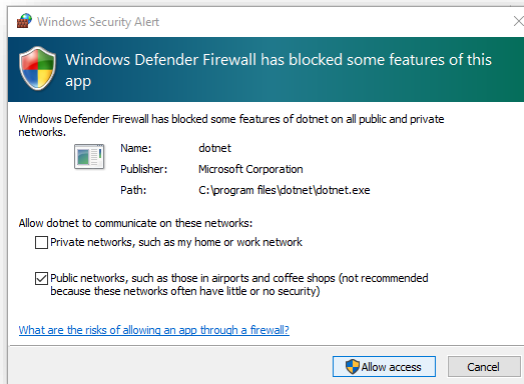
1. Open the `DC_EMV_Server` solution.

Tip: Unload the demo client and perso app projects to speed up solution loading.

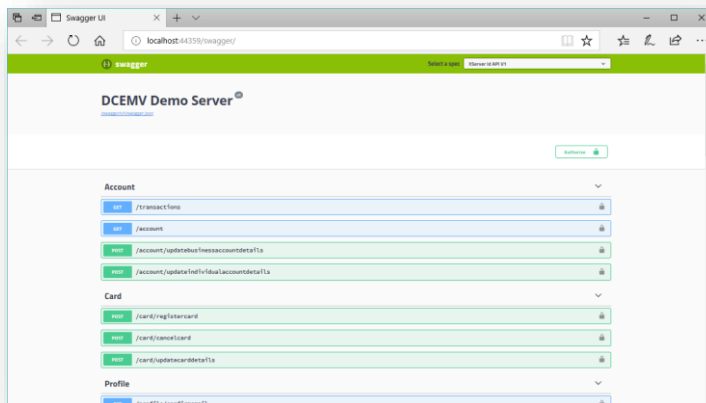
2. To run the server in standalone set the following:



3. If the server starts successfully, the URL configured in the project settings debug options, synced with the `launchSettings.json` file, will be started up, in this case the swagger page.
4. If you get prompted to allow access through the firewall, click allow access.



5. This is the swagger page, generated by the Swahbuckle library integrated into the DCEMV_DemoServer application.



7.4 Run Demo App

Before running the Demo App make sure terminal configuration is added to support the card applet AID.

The card applet uses AID A000000050010101 and its equivalent terminal configuration for transaction type PurchaseGoodsAndServices and using Kernel3 is shown below. This can be found in CodeData.cs of DCEMV_ConfigurationManager.

```
<TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <TTQAsBytes>9F660427C04000</TTQAsBytes>
    <KernelEnum>Kernel3</KernelEnum>
    <RIDEnum>A000000050</RIDEnum>

<TransactionTypeEnum>PurchaseGoodsAndServices</TransactionTypeEnum>
    <StatusCheckSupportFlag>>false</StatusCheckSupportFlag>
    <ZeroAmountAllowedFlag>>true</ZeroAmountAllowedFlag>

<ReaderContactlessTransactionLimit>100000</ReaderContactlessTransactionLimit>
    <ReaderContactlessFloorLimit>30000</ReaderContactlessFloorLimit>
    <TerminalFloorLimit_9F1B>20000</TerminalFloorLimit_9F1B>
    <ReaderCVMRequiredLimit>10000</ReaderCVMRequiredLimit>

<ExtendedSelectionSupportFlag>>false</ExtendedSelectionSupportFlag>

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
</TerminalSupportedContactlessKernelAidTransactionTypeCombination>
```

In the constructor of the SessionSingleton class of the DCEMV_DemoApp are a few hard-coded values for username and password, this is for convenience in order to not have to retype the values each time the app is being tested.

Connectivity settings are as follows:

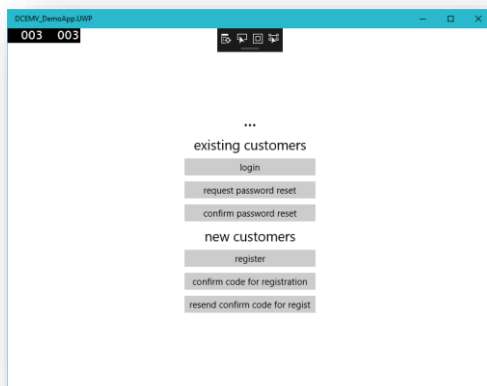
Client configure in	Server Type	Svr/DB Configured in	Default for Server	Default for DB
SessionSingleton	Standalone	launchSettings.JSON	http://localhost:44359	localhost
SessionSingleton	Docker	docker-compose.yml	https://192.168.0.100	192.168.0.100

We are now ready to test using either the card simulator or a physical card.

6. Open the DC_EMV_Demo_App solution in a new instance of Visual Studio.

Tip: Unload the demo server projects to speed up solution loading.

7. You will now have 2 instances of Visual Studio open, one running the server and this one. If using the card applet simulator, you will also have a copy of IntelliJ open running the DCEMVCardEmulator. Ensure each solution in each Visual Studio instance is built before running the applications, if there are pending changes in one that need to be built and the other is running its application, the first may not be able to build the solution as some of the required files may be locked by the running instance.
8. Run the app, the first screen is shown blow.



9. Click register a user

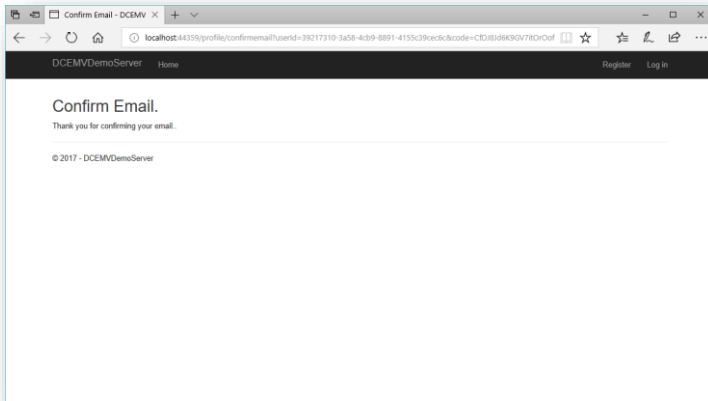
The screenshot shows a 'Register' form window titled 'DCEMV_DemoApp.UWP'. The form has three input fields: an email field containing 'testuser@domain.com', a password field with masked characters, and a confirm password field also with masked characters. To the right of the email field is a green message 'The email address is valid'. To the right of the password field is a green message 'The password meets the requirements'. To the right of the confirm password field is a green message 'The passwords match'. Below these fields, a list of requirements is shown: 'The password must forefill the following requirements: 1) at least 1 upper case character, 2) at least 1 lower case character, 3) at least 1 numerical character, 4) at least 1 special character. The length must be between 6 and 10 characters'. At the bottom are 'ok' and 'cancel' buttons.

10. Click OK. If successful you should see the following:

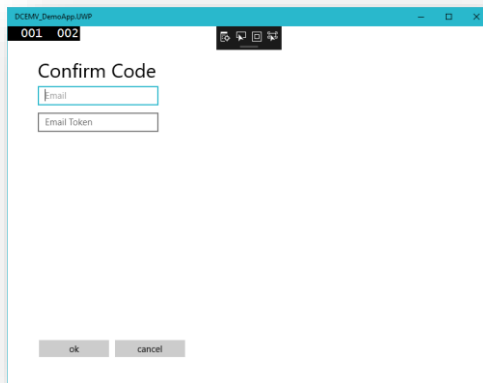
The screenshot shows the same 'Register' form window, but the email field now contains a placeholder URL 'http://domain.com/307633'. To the right of this field is a red message 'The email address is invalid'. The password and confirm password fields remain masked, and the messages 'The password meets the requirements' and 'The passwords match' are still present in green. The list of requirements and the 'ok'/'cancel' buttons are also visible.

To verify the email address; The verification URL is placed in the email field, navigate to this URL in your browser to verify the email address. In reality an email would be send to the email address containing the URL, SendGrid is implemented as mentioned in the

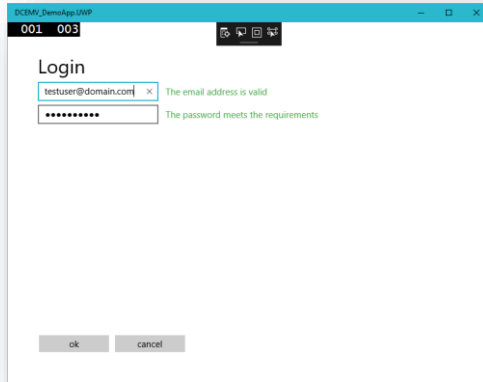
11. Run Server App section.



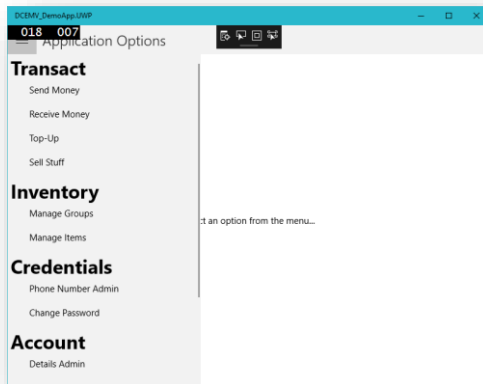
12. The email address can also be confirmed by pasting the code portion of the URL into the Confirm Code View

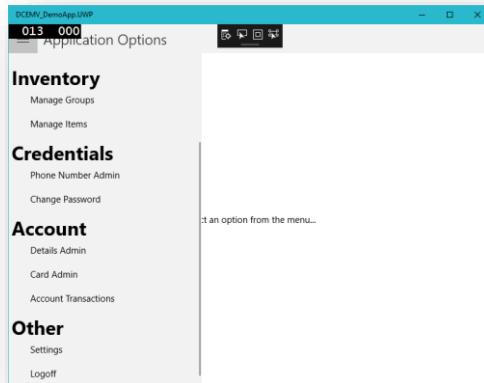


13. On the main screen click Login

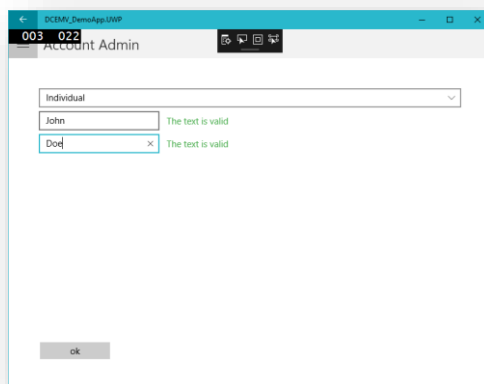


14. After logging in the following options are available



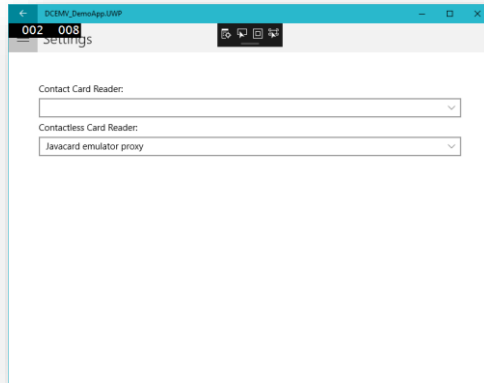


15. The account details must be completed before the functions can be used

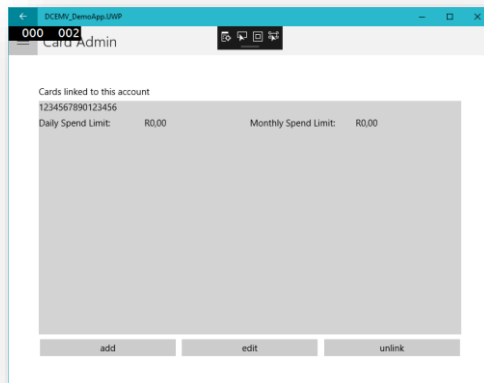
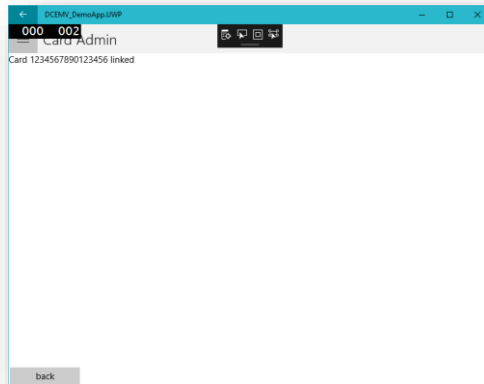


16. Select the card reader in settings, if using the emulator select the emulator for contactless, if you wish to test Top Up with a contact card, you will need a contact card reader connected to your system. Select that card reader for contact. At least one card reader must be selected in this view to enable the other

functions. The default setup for the Demo Windows version of the app is no card reader for contact and the Emulated card for contactless.



17. Register a card with the logged in apps account. If running the simulator only a single simulated card can be used. Since the default for the Windows Demo App is the emulator for contactless, you can register this card by clicking the Add Card button.



18. Once the card is linked, select it and click edit. Change the daily spend limits.

The screenshot shows a web application window titled "DC EMV Demo App" with a sub-header "Card Admin". The main heading is "Update Card". There are three input fields, each with a green validation message to its right:

- Text Card: The text is valid
- 10000: The amount is valid
- 10000: The amount is valid

At the bottom of the form are two buttons: "ok" and "cancel".

The screenshot shows the same "DC EMV Demo App" window, now displaying a table of linked cards. The table has two columns: "Test Card" and "Monthly Spend Limit". The first row shows a card number "1234567890123456" and a limit of "R100.00". Below the table are three buttons: "add", "edit", and "unlink".

Test Card	Monthly Spend Limit
1234567890123456	R100.00

19. A top must be done to add funds to the account, or transactions cannot be done. The demo allows the DC EMV card to be used to top up the account, in reality this would not be allowed. An EMV card can also be used.

←DCIMV_DemoApp (WP)

003002top:top

100000

x

The amount is valid

Next

←DCIMV_DemoApp (WP)

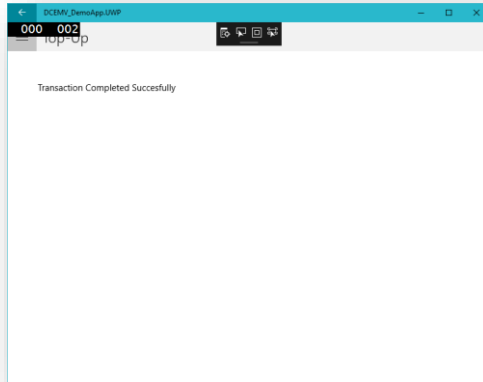
001002top:top

Please Tap or Insert the card you wish to make the payment with.

R100,00

Present Card

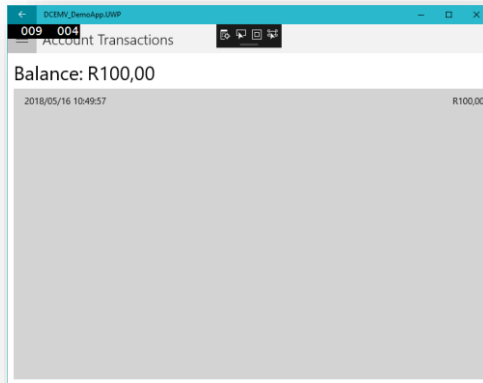
Cancel



20. The backend has been configured to accept the Top Up request, if the EMV contactless card returns Approved or Request an Online Approval. In reality one would not allow DC EMV cards to be used for Top Up. Since the default configuration is to use the Emulated DC EMV card, this will be used for the Top Up, since it is an EMV compatible card app. It will request Online Approval and since the backend is configured to treat online approval requests as approvals it will process the Top Up. If using a physical card reader, and both contact and contactless interfaces are selected in the Settings view then a DC EMV card or a contact or contactless EMV card can be used. If a contact EMV card is used, and it requests Online Approval during 1st gen AC, the terminal will call the backend AuthTransactionToIssuer and if it cannot go online via the configured SPDH Approver the terminal will revert back to the card which in turn may choose to approve or decline the transaction. If approved the backend Top Up function will be called and a Top Up processed. In a production environment the backend would only process a

Top Up from an EMV card and only if the card/online approver approves the transaction.

21. The account transactions view can be used to check your balance and the transactions done on the account



The Demo app also has a basic POS view, and Inventory Management views.

22. Add an Inventory Group (Category)

DCIMV_DemoApp UWP

002 002

Inventory Group Admin

Add/Update Inventory Group

Tech Books The text is valid

All IT Books x The text is valid

ok cancel

DCIMV_DemoApp UWP

000 002

Inventory Group Admin

Name: Tech Books Description: All IT Books

add edit remove

23. Add an Inventory Item

DCMV_DemoApp UWP

Inventory Item Admin

Add/Update Inventory Item

User Manual The text is valid

DC EMV Handbook The text is valid

12345 The number is valid

1000 The amount is valid

Tech Books The selected item is valid

ok cancel

DCMV_DemoApp UWP

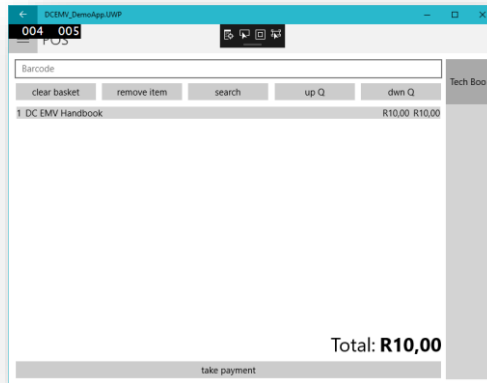
Inventory Item Admin

Search String

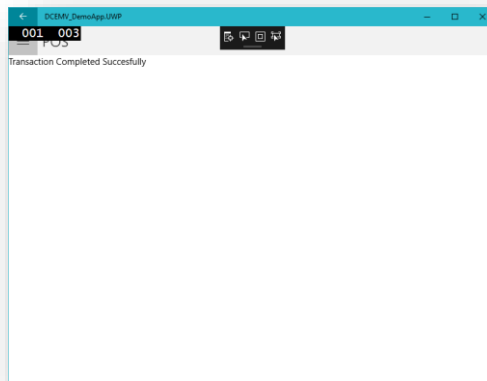
Name: User Manual	Description: DC EMV Handbook
Barcode: 12345	Price: R10,00 Group: Tech Books

add edit remove

24. The sell stuff screen allows the creation of a basket and then the taking of a payment for that basket of items. Add the item just added to the basket, by typing in its barcode and pressing enter, or searching for it using the Search Text box or by navigating the categories and items within those categories with the buttons on the right.



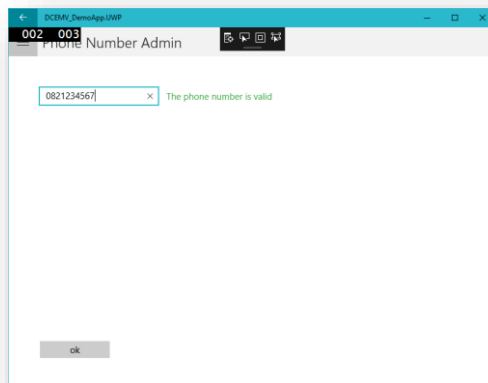
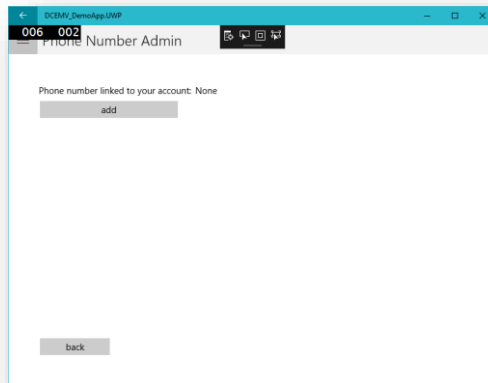
25. Click the take payment button, present your card or wait for the emulator to interact. Once complete the following view will be shown.



26. Open the Account Transactions View to view the transactions. The items sold during the transaction are also saved in the database. This view could be modified to view the items

included in the POS transaction for both the buyer and the seller.

27. A phone number can be added to the account and then verified via Twilio. Open the Phone Number Admin view and fill in the mobile number.

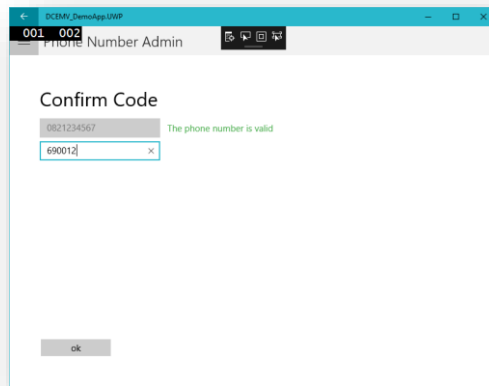


28. In the output window of the debugger will be the following message:

DCEMV.DemoServer.Components.AuthMessageSender:Information:
SMS: +27821234567, Message: Your DCEMV OTP is: 690012

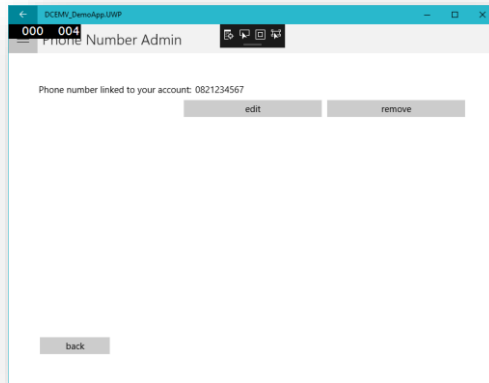
Or if Twilio support is enabled, a SMS will be send to the phone number with the OTP in the message.

29. Enter the OTP in the Confirm Code view.

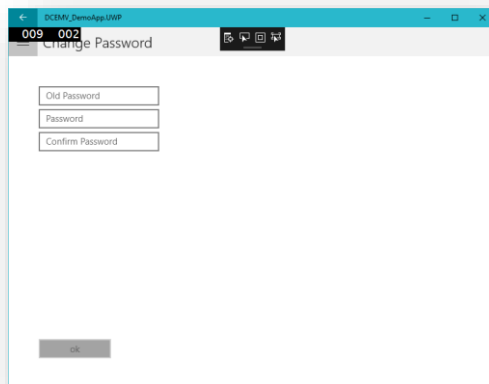


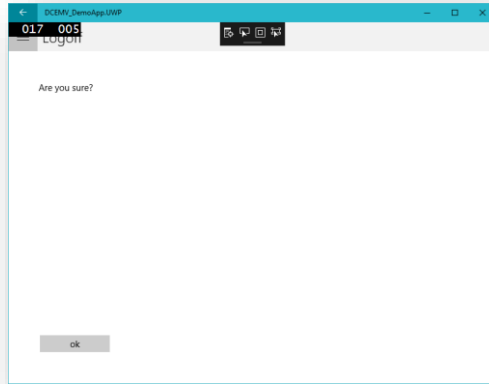
The screenshot shows a mobile application interface titled "Confirm Code". At the top, there's a header bar with "PHONE Number Admin" and some icons. Below the header, the title "Confirm Code" is displayed. Underneath, there's a text input field containing the phone number "0821234567". To the right of this field, a green message states "The phone number is valid". Below the phone number field, there's another text input field containing the OTP "690012". At the bottom of the dialog, there is an "OK" button.

30. The verified phone number is now registered with the app. It can be edited, requiring another confirmation or removed from the account.



31. There are also a change password and logoff view



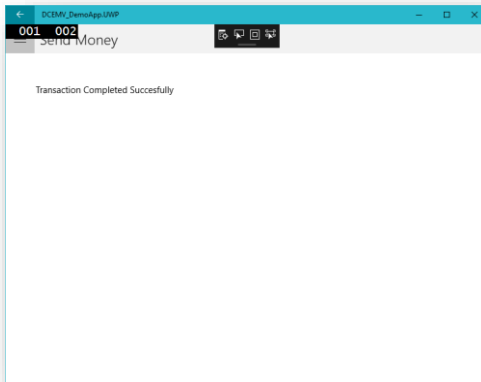
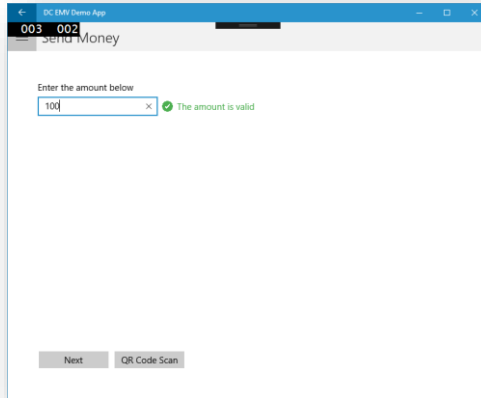


7.4.1 Transactions with Cards or HCE

32. Transactions can now be done (if using the simulator or if using the same card as registered on the app, transactions will happen on the same account, i.e. from you to you or to you, from you)

7.4.1.1 Send Money

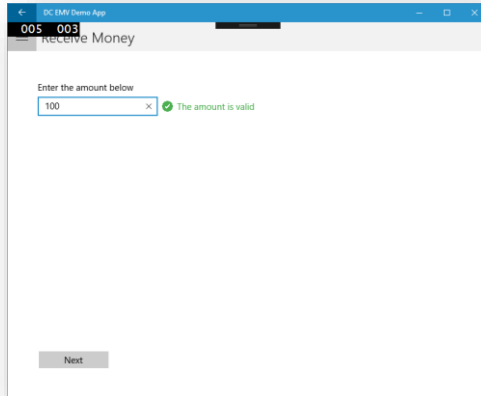
33. To send money from your account to their card, use the Send Money View. Since the card we will use is also linked to your account, the money will transfer from your account back to your account.
34. Enter the amount and press next. Tap the card or a phone with the app installed (e.g. Android phone with the DECEMV_Demo configured as the default payment app in the Android Tap and Pay settings screen)



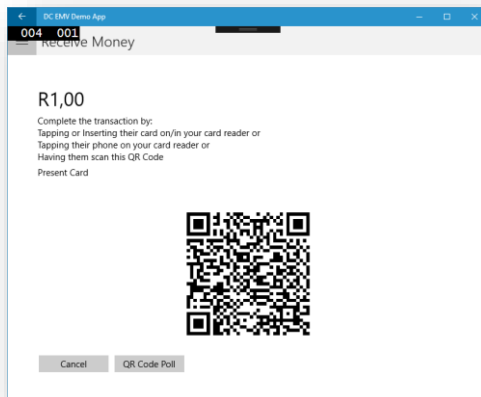
35. Open the Account Transactions View to view the transactions.

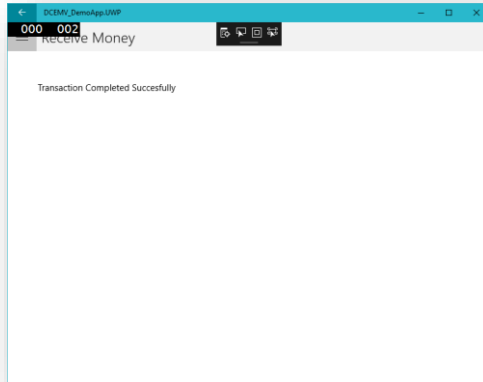
7.4.1.2 Receive Money

36. To send money from your card to their account, use the Receive Money View. Since the card we will use is also linked to your account, the money will transfer to your account back from your account.



37. Tap the card or a phone with the app installed (e.g. Android phone with the DECEMV_Demo configured as the default payment app in the Android Tap and Pay settings screen)



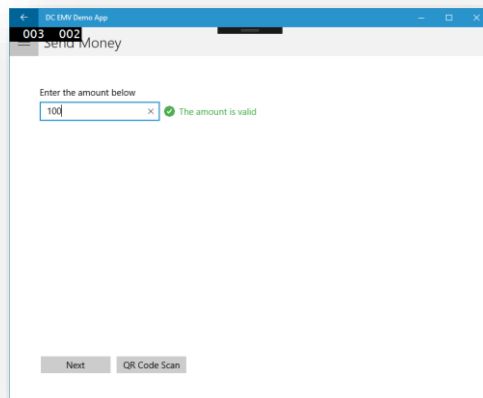


38. Open the Account Transactions View to view the transactions.

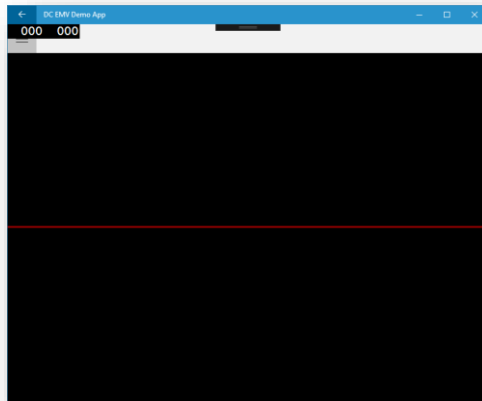
7.4.2 Transactions with QR Codes

7.4.2.1 Send and Receive Money

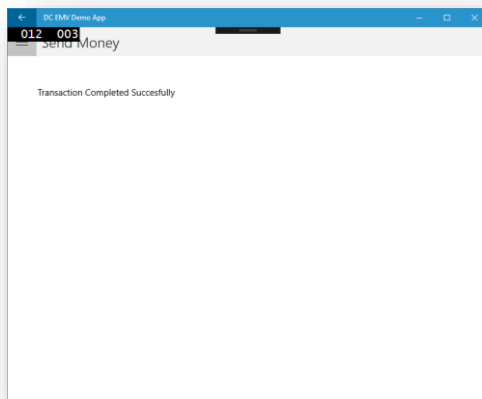
39. Use the Send Money option, but this time press the scan QR code option.



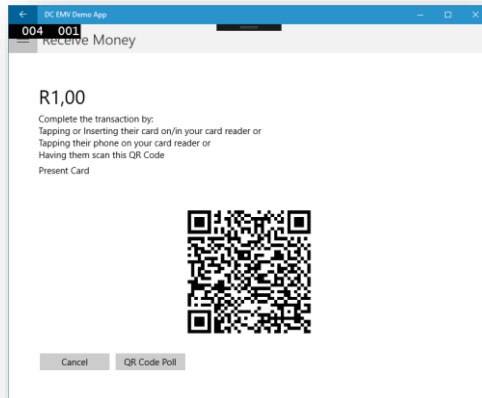
40. Scan the QR code presented on another instance of the application.



41. Once scanned the app will use the configured online approver to process the transaction, if successful the view below will display



42. In order to present the barcode, install the DCEMV_DemoApp on another device and select the send money option.
43. Enter the amount and press the next button. The QR code will be presented on the next view.



44. Once the scanner of the barcode has completed the transaction, press the QR Code poll button to see if the transaction was completed and if it was approved or declined.

7.5 Run Personalization App

7.5.1 Understanding Card Keys

The first step in personalizing the DC EMV Card Applet is understanding card keys and how they are generated. This key is a 3DES Key. This key is normally generated in a way that will allow a server receiving a transaction from this card to interact with an HSM which will recreate the key and use it to verify the cryptogram the card has sent. This means the key must be generated by an HSM that has the same root keys as the HSM which will eventually be used to verify the cryptogram. DC EMV uses an in memory HSM, loaded with root keys found in a file deployed with the library. Obviously this HSM and its root keys must be secured in an acceptable way in a production environment.

A card key is created by using the Master Application Cryptogram key, the card PAN, the card PAN sequence number and applying an algorithm that creates the card key.

The Master Application Cryptogram key is generated on the HSM by specifying a key variant and scheme. The key is created and encrypted. This key can then be stored in the clear on any server, as it is encrypted.

When an HSM needs to validate a cryptogram:

- it uses the master ac key, decrypted from the encrypted master ac key, and from it derives a card key, or uses it as is, depending on the scheme.
- it uses the pan, the pan sequence number, the application transaction counter (optional), the UPN (optional) and the data used by the card for this transaction when it calculated the cryptogram.
- using the key calculated in the first step, it calculates the transaction cryptogram

- If the calculated cryptogram matches the cryptogram sent by the card then validation has passed.

7.5.2 Understand the Perso File

The xml personalization file must be created/updated for the card. The PersoEMV.xml file in DCEMV_GlobalPlatformProtocol can be used as a sample. The values specified in this file must be the same used as when the key for the card was calculated. The values here are the same used as those used in doTestPerso when running the simulator.

The CardMasterKey and Security Domain AID must be supplied by the card manufacturer. It is used by the perso app to access the globalplatform security applet loaded on the card by the manufacturer.

The SessionSingleton constructor must be updated with the Master Key and the AID, this is used to populate the respective values in the Perso Views. The Perso.xml file contains an element to populate the Master Key, however this is not currently used, the master key entered on the Perso Views is used to Auth with the card.

The correct card key must be selected in the perso file, depending on whether the simulated payment provider is to be used with DCEMV_DemoEMVApp or if the DC EMV Demo server, using the HSM, is to be used.

The structure of the file is dependent on the Applet writer and how they chose to store their perso data when designing the Applet. Since I wrote the applet I also determined the perso structure. The GlobalPlatform will, via the security domain applet, call the install method on the Applet, passing it the data in ADPU store data commands. The applet will then unpack these commands and providing the data has been formatted the way the applet expects, will then store the data. This is a one-time process, however the

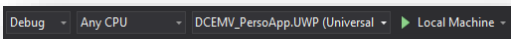
DCEMVCARD applet has not enforced this rule, so a card can be personalized multiple times.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<perso xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.xserver.com/perso"
  xsi:schemaLocation="http://www.xserver.com/perso Perso.xsd">
  <keys>
    <key Type="CardMasterKey" Value="12345678901234567890123456789012"/>
  </keys>
  <application type="Credit/Debit No SDA/DDA/CDA">
    <commands>
      <install ExecutableLoadFileAID="A0000000050"
ExecutableModuleAID="A00000000500101" ApplicationAID="A0000000050010101">
        <tokens/>
      </install>
      <storeData DGI="8001" Description="3DES Key Contactless">
        <data>8CB9F7D54362B0A240D0AE626780A86B</data>
      </storeData>
      <storeData DGI="0101" Description="DGI by SFI and REC">
        <tlvxml>
          <tag Name="70" Description="Store Data Template" />
          <children>
            <tlvxml>
              <tag Name="5A" Description="Application Primary Account Number
(PAN)" />
              <value Value="1234567890123456" />
            </tlvxml>
            <tlvxml>
              <tag Name="5F34" Description="Application Primary Account
Number (PAN) Sequence Number" />
              <value Value="01" />
            </tlvxml>
          </children>
        </tlvxml>
      </storeData>
      <storeData DGI="0102" Description="DGI by SFI and REC">
        <tlvxml>
          <tag Name="70" Description="Store Data Template" />
          <children>
            <tlvxml>
              <tag Name="57" Description="Track 2 Equivalent Data" />
              <value Value="1234567890123456D2512201" />
            </tlvxml>
          </children>
        </tlvxml>
      </storeData>
    </commands>
  </application>
</perso>
```

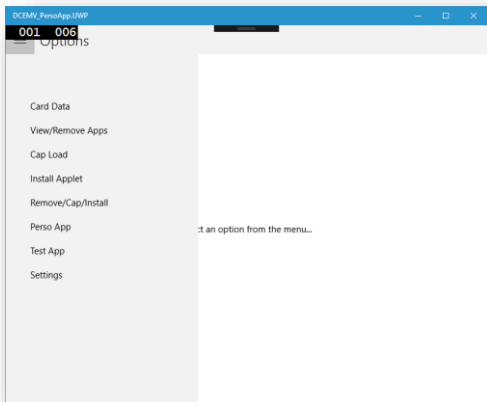
The personalization app is then used with the xml file to personalize the card payment applet installed previously.

7.5.3 Perso the Card

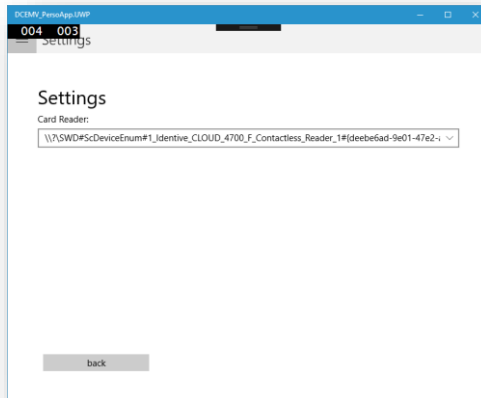
1. To run the Personalization Application set the start-up project as follows:



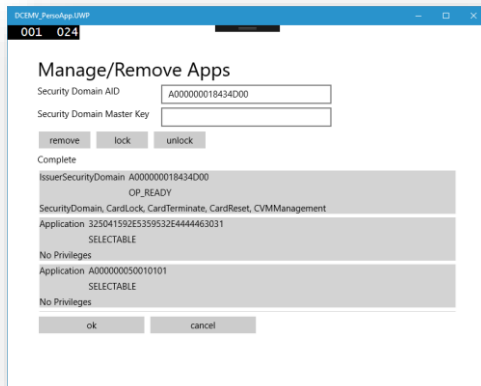
2. Once running the following options are available:



3. Select the card reader you plan to use; currently only contactless card readers are supported as the DC EMV Card Applets are built to be used with a contactless only card.



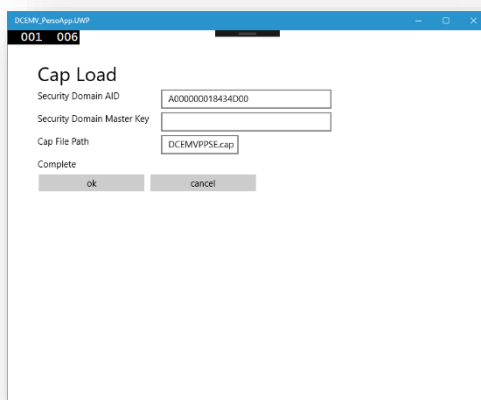
4. The get card data view will get general card data from the card. This step can be skipped.
5. The Manage/Remove apps will query the card for all the apps installed on the card. This view can be used to view, add, lock and remove existing apps.



The Security Domain AID and Security Domain Master Key must be supplied by the manufacturer/supplier of the card. The card shown in the screenshot already has the PPSE applet and the EMV DC Card Applet cap files loaded and the applet installed. The first item in the list is the security domain applet that will be preinstalled on the card. It must be global platform compatible. The card used in this tutorial supports GP 2.1.1 / VGP 2.1.1 and JavaCard 2.2.1. The second item in the list is the PPSE applet and the third is the DC EMV Card Applet. Both have been installed and made selectable.

The steps following will describe the perso process required to get a blank card into the state shown above.

6. Place the 2 cap files generated by IntelliJ into the DCEMV_PersoApp.UWP\bin\x86\Debug\AppX folder or put the full folder path in the path text box of the view.
7. Load the DCEMVPPSE.cap file first. Press OK, place the card in the card reader field, and once complete the view will display “Complete”.



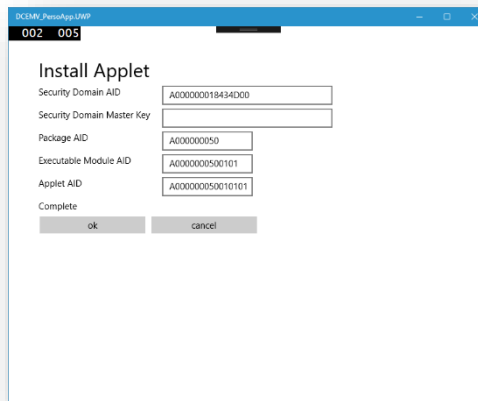
8. Then the DCEMVCard.cap

The screenshot shows a dialog box titled "DCEMVPPSE Applet" with a status bar at the top displaying "004 003". The main title is "Cap Load". It contains four text input fields: "Security Domain AID" with the value "A000000019434D00", "Security Domain Master Key" (empty), "Cap File Path" with the value "DCEMVCard.cap", and "Complete" (empty). At the bottom are "ok" and "cancel" buttons.

9. Now we will install an applet instance for the cap files just loaded. First the DCEMVPPSE applet instance. The values loaded in the text boxes can be found in the constructor of each respective view.

The screenshot shows a dialog box titled "DCEMVPPSE Applet" with a status bar at the top displaying "002 004". The main title is "Install Applet". It contains five text input fields: "Security Domain AID" with the value "A000000019434D00", "Security Domain Master Key" (empty), "Package AID" with the value "A000000060", "Executable Module AID" with the value "A0000000600101", and "Applet AID" with the value "325041592E5359532E4444463031". At the bottom are "ok" and "cancel" buttons.

10. Now the DCEMVCard applet instance.



11. At this point both applets are installed and selectable however the DCEMVCard applet still needs to be personalized.

The Remove/Cap/Install view will remove the DCEMVCard applet and cap file, then load the cap file and Install the applet instance, this is useful during debugging, as it is quicker than doing the steps individually.

12. Place the PersoEMV.xml file found in the
DCEMV_GloablPlatformProtocol\Model in the
DCEMV_PersoApp.UWP\bin\x86\Debug\AppX folder.

13. Open the XML Perso view and click ok.

Perso

Security Domain AID: A000000018434D00

Security Domain Master Key:

XML Path: PersoEMV.xml

Perso Complete

ok Cancel

14. The card is now ready to use.

15. The test EMV View will attempt a transaction on the card. If it works you will see the returned EMV tags as follows:

Test App

Applet AID: A000000050010101

Complete

T:[82] Application Interchange Profile L:[2]	V:[0000]
T:[57] Track 2 Equivalent Data L:[12]	V:[123456789012345602512201]
T:[5F20] Cardholder Name L:[2]	V:[/]
T:[5A] Application Primary Account Number (PAN) L:[8]	V:[1234567890123456]
T:[5F34] Application Primary Account Number (PAN) Sequence Number L:[1]	V:[01]
T:[9F10] Issuer Application Data L:[32]	V:[00000000000000000000000000000000]
T:[9F26] Application Cryptogram L:[8]	V:[826FA4212C4325FD]
T:[9F27] Cryptogram Information Data L:[1]	V:[00]
T:[9F36] Application Transaction Counter (ATC) L:[2]	V:[0001]
T:[9F6C] Invalid tag L:[2]	V:[2800]
T:[9F6E] Third Party Data L:[4]	V:[00000000]
T:[9F7C] Merchant Custom Data L:[32]	V:[00000000000000000000000000000000]

ok cancel

16. The card personalization is complete and the card can now be used with the DCEMV Demo app.

7.5.4 Re-Perso the Card

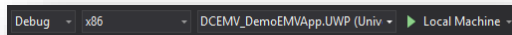
In order to re-perso the card, first remove the loaded cap file, this will remove the cap file and any app installed from that cap file. The manage / remove apps view will display both the installed apps and the loaded cap files.

7.6 Demo Application for EMV Contact and Contactless Kernels

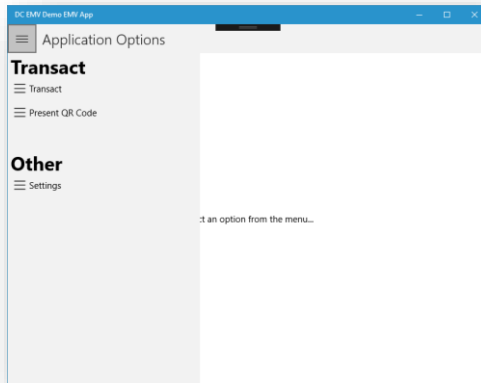
The DCEMV_DemoEMVApp is provided in order to be able to test the contact and contactless kernels with EMV cards. It is a quick way to test the kernels without requiring the server to be configured and running.

Note that for all views the code logs detailed messages to the console. These can be very useful when testing the kernel or if trying to figure out why an EMV card is behaving the way it is.

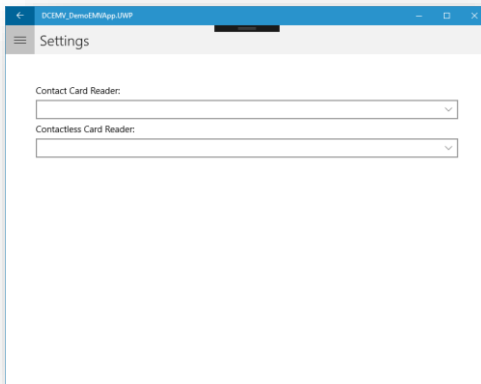
1. In the constructor of MainPage, for Windows, or MainActivity, for Android, ensure the online approver is configured correctly with the IP address of the switch that is going to process the transaction. This approver is only used for contact cards who require Online Approval after 1st Gen AC.
2. To run this app set the DCEMV_DemoEMVApp.UWP as the start-up project (or DCEMV_DemoEMVApp.android for Android) and then run the app.



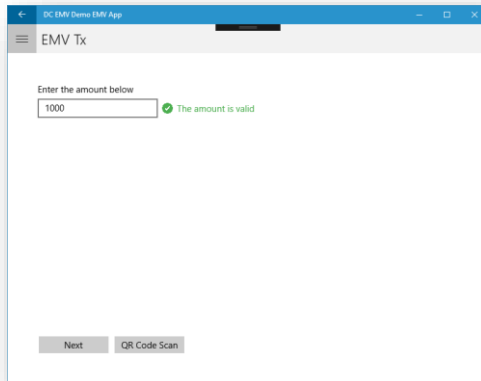
3. The following options are available:



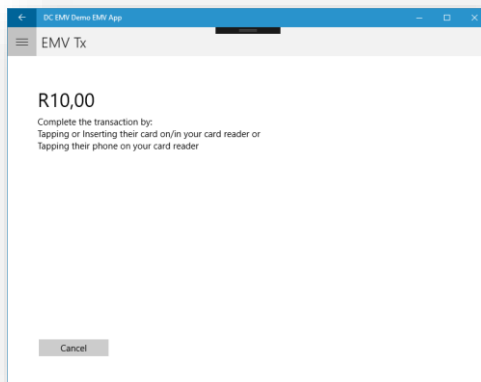
4. Select the card readers to be used.



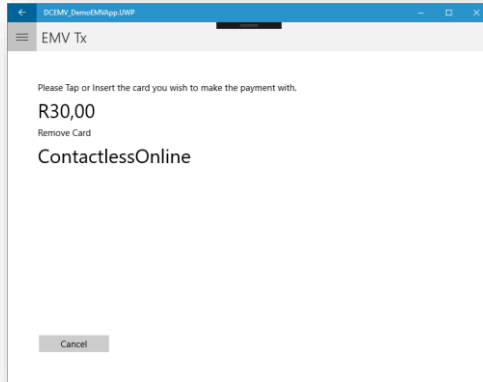
5. Do a transaction. Select the Transact option. Select an amount and click next.



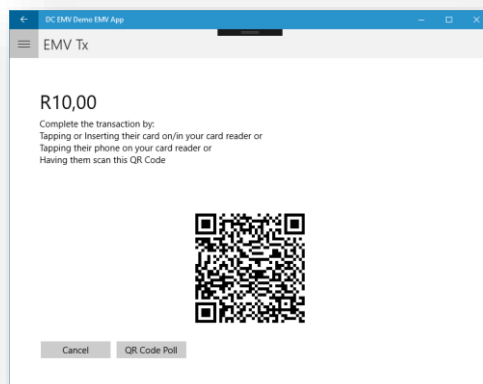
6. Tap (Card or HCE) or insert the card you want to use.



7. The final view shows the outcome of the transaction.



8. Here a DC EMV card was used, but any EMV contact or contactless card be used that works with the kernels implemented. See Sample Terminal Application for the detail logged to the console.
9. To do a QR code transaction use the transact option and scan a QR code presented by another instance of the app.
10. The Present QR code option can be used to present a QR code.



7.7 Running the apps on Android

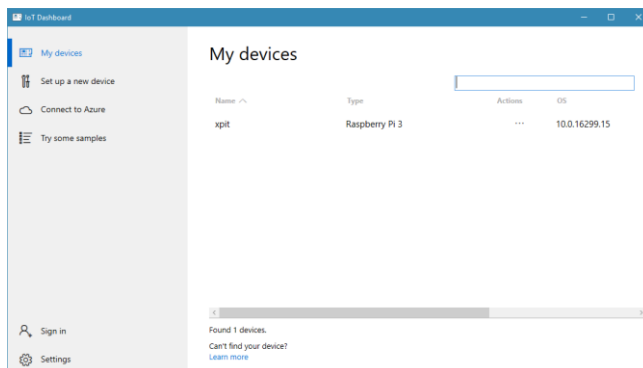
This apps were tested on a Samsung SM-T365 Tablet with Android 5.1.1. This device has built in NFC. The ACS1255 was tested with the Samsung Tablet. The apps were also tested on a Nexus 5 phone with Android 6.0.1 with its built-in NFC. The apps were tested on the Android devices tethered to a PC running Visual Studio.

7.8 Running the apps on the Raspberry Pi

The hardware setup for this was a Raspberry Pi 3 with the OM5577 module and the Raspberry Pi 7-inch touch screen.

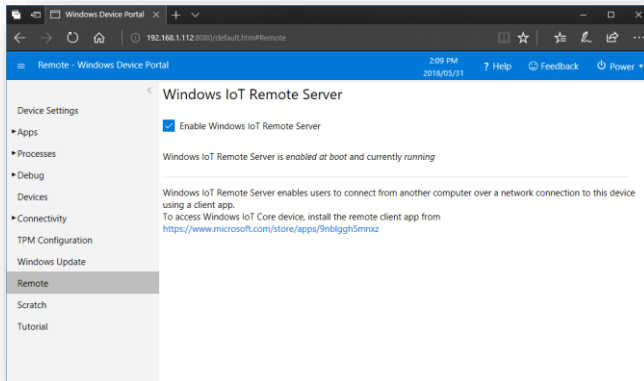
Windows IOT was setup / run on the Pi using the Windows IOT core dashboard. If you have no Raspberry Pi touch screen the Windows remote IOT client can be used. Version 10.0.16299.15 of Windows IOT was used. To configure the Pi I do the following:

1. Use the dashboard to flash the SD card of your Pi with the OS

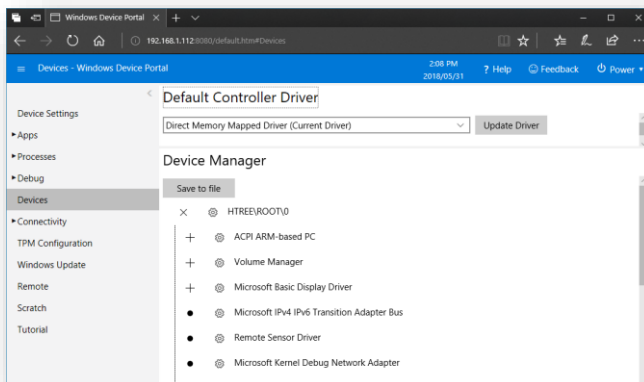


2. Plug the Pi's ethernet port into the same network as your PC
3. Start the Pi
4. Start the Windows IOT core dashboard
5. Once the Pi is discovered, right click on it select Open in Device Portal

6. Configure the Wi-Fi in the portal
7. Once the Wi-Fi is connected, the Wi-Fi can be used rather than Ethernet
8. Enable Windows IOT remote server in the Portal



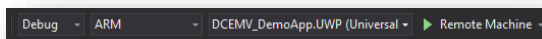
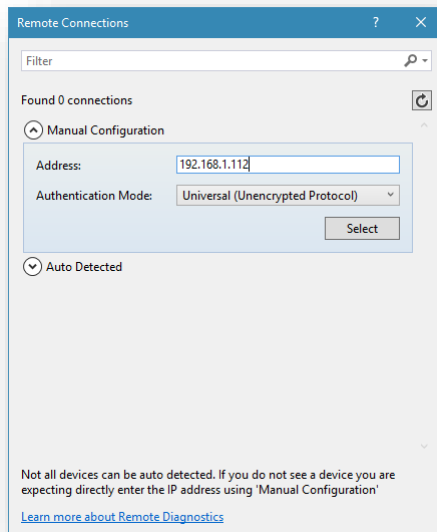
9. Enable Direct Memory Mapped Driver in the Default Controller Driver page



10. Use the IOT remote client if there is no display connected to the Pi

Enable the CPU config will add the correct conditional compilation symbol in each app project file in Visual Studio. For ARM it will add the Pi conditional compilation symbol which switched the code that loads the correct card reader drivers.

Select the ARM and Remote Machine option on the Visual Studio debug toolbar. If prompted to select the remote machine, select the Pi IP address from the list or add the IP manually.



Tip: When switching CPU architecture, do a rebuild of the project.

8 Sample Terminal Application Log

This is a copy of the trace produced for an EMV transaction, in this case a DC EMV transaction, however, EMV card transactions are just as detailed.

```
[2018/05/16 15:10:00.579] [PublicKeyCertificateManager]
CA Public Key Certificates:

'DCEMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.
[2018/05/16 15:10:01.129] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfCAPublicKeyCertificate
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <CAPublicKeyCertificate>
      <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
      <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>A90FCD55AA2D5D9963E35ED0F440177699832F49C6BAB15CDAE5794BE93F934D4462
D5D12762E48C38BA83D8445DEAA74195A301A102B2F114EADA0D180EE5E7A5C73E0C4E11F67A4
3DDAB5D55683B1474CC0627F44B8D3088A492FFAADAD4F42422D0E7013536C3C49AD3D0FAE964
59B0F6B1B6056538A3D6D44640F94467B108867DEC40FAAECDD740C00E2B7A8852D</Modulus>
      <Exponent>03</Exponent>
      <ExpiryDate/>
      <Index>250</Index>
      <ActivationDate />
      <Checksum />
      <RID>A000000004</RID>
    </CAPublicKeyCertificate>
  </ArrayOfCAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>CBCBDB21CCE3A3369EC6FFAB4C26D8F8660FBC75DBA42115F9EDB4E3D654AAF2434C
CEB57837449D0A0A59683D3A199F2FC36149951EBAA192D1336A043016BE72F443B1A4B636CF7
A464F59E480B79C0A8A68684822278B0922C4800931DEC8C73F58B2DE8B2EBA4203AD62191473
8AB49EF5653A6873AC4216E8127D275DB9E4A9A848CDF85B5BD4719F19A33EF953</Modulus>
      <Exponent>03</Exponent>
      <ExpiryDate>1220</ExpiryDate>
      <Index>173</Index>
      <ActivationDate />
      <Checksum />
      <RID>A000000004</RID>
    </CAPublicKeyCertificate>
  </ArrayOfCAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>
```

<Modulus>A89F25A56FA6DA258C8CA8B40427D927B4A1EB4D7EA326BBB12F97DED70AE5E4480F
C9C5E8A972177110A1CC318D06D2F8F5C4844AC5FA79A4DC470BB11ED635699C17081B90F1B98
4F12E92C1C529276D8AF8EC7F28492097D8CD5BECEA16FE4088F6CFAB4A1B42328A1B996F9278
B0B7E3311CA5EF856C2F888474B83612A82E4E00D0CD4069A6783140433D50725F</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1217</ExpiryDate>
<Index>7</Index>
<ActivationDate />
<Checksum>B4BC56CC4E88324932CBC643D6898F6FE593B172</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>16</PublicKeyAlgorithmIndicator>

<Modulus>D9FD6ED75D51D0E30664BD157023EAA1FFA871E4DA65672B863D255E81E137A51DE4
F72BCC9E44ACE12127F87E263D3AF9DD9CF35CA4A7B01E907000BA85D24954C2FCA3074825DDD
4C0C8F186CB020F683E02F2DEAD3969133F06F7845166ACEB57CA0FC2603445469811D293BFEF
BAFAB57631B3D091E796BF850A25012F1AE38F05AA5C4D6D03B1DC2E568612785938BBC9B3CD3
A910C1DA55A5A9218ACE0F7A21287752682F15832A678D6E1ED0B</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1224</ExpiryDate>
<Index>8</Index>
<ActivationDate />
<Checksum>20D213126955DE205ADC2FD2822BD22DE21CF9A8</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>9D912248DE0A4E39C1A7DDE3F6D2588992C1A4095AFBD1824D1BA74847F2BC4926D2
EFD904B4B54954CD189A54C5D1179654F8F9B0D2AB5F0357EB642FEDA95D3912C6576945FAB89
7E7062CAA44AA06B8FE6E3DBA18AF6AE3738E30429EE9BE03427C9D64F695FA8CAB4BFE3768
53EA34AD1D76BFCAD15908C077FFE6DC5521ECEFD5D278A96E26F57359FFAEDA19434B937F1AD9
99DC5C41EB11935B44C18100E857F431A4A5A6BB65114F174C2D7B59FDF237D6BB1DD0916E644
D709DED56481477C75D95CDD68254615F7740EC07F330AC5D67BCD75BF23D28A140826C026DBD
E971A37CD3EF9B8DF644AC385010501EFC6509D7A41</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1225</ExpiryDate>
<Index>9</Index>
<ActivationDate />
<Checksum>1FF80A40173F52D7D27E0F26A146A1C8CCB29046</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>996AF56F569187D09293C14810450ED8EE3357397B18A2458EFAA92DA3B6DF6514EC
060195318FD43BE98BF0CC669E3F844057CBDDF8BDA191BB64473BC8DC9A730DB8F6B4EDE3924
186FFD9B8C7735789C23A36BA0B8AF65372EB57EA5D89E7D14E9C7B6B557460F10885DA16AC92

3F15AF3758F0F03EBD3C5C2C949CBA306DB44E6A2C076C5F67E281D7EF56785DC4D75945E491F01918800A9E2DC66F60080566CE0DAF8D17EAD46AD8E30A247C9F</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1250</ExpiryDate>
<Index>146</Index>
<ActivationDate />
<Checksum>429C954A3859CEF91295F663C963E582ED6EB253</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>ACD2B12302EE644F3F835ABD1FC7A6F62CCE48FFEC622AA8EF062BEF6FB8BA8BC68B
BF6AB5870EED579BC3973E121303D34841A796D6DCBC41DBF9E52C4609795C0CCF7EE86FA1D5C
B041071ED2C51D2202F63F1156C58A92D38BC60BDF424E1776E2BC9648078A03B36FB554375FC
53D57C73F5160EA59F3AFC5398EC7B67758D65C9BFF7828B6B82D4BE124A416AB7301914311EA
462C19F771F31B3B57336000DF732D3B83DE07052D730354D297BEC72871DCCF0E193F171ABA
27EE464C6A97690943D59BDABB2A27EB71CEEBDFAA1176046478FD62FEC452D5CA393296530AA
3F41927ADFE43A2DF2AE3054F8840657A26E0FC617</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1250</ExpiryDate>
<Index>148</Index>
<ActivationDate />
<Checksum>C4A3C43CCF87327D136B804160E47D43B60E6E0F</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>BE9E1FA5E9A803852999C4AB432DB28600DCD9DAB76DFAAA47355A0FE37B1508AC6B
F38860D3C6C2E5B12A3CAAF2A7005A7241EBAA7771112C74CF9A0634652FBCA0E5980C54A6476
1EA101A114E0F0B5572ADD57D010B7C9C887E104CA4EE1272DA66D997B9A90B5A6D624AB6C57E
73C8F91900EB5F684898EF8C3DBEFB330C62660BED88EA78E909AFF05F6DA627B</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1250</ExpiryDate>
<Index>149</Index>
<ActivationDate />
<Checksum>EE1511CEC71020A9B90443B37B1D5F6E703030F6</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>98F0C770F23864C2E766DF02D1E833DFF4FFE92D696E1642F0A88C5694C6479D16DB
1537BFE29E4FDC6E6E8AFD1B0EB7EA0124723C333179BF19E93F10658B2F776E829E87DAEDA9C
94A8B3382199A350C077977C97AFF08FD11310AC950A72C3CA5002EF513FCCC286E646E3C5387
535D509514B3B326E1234F9CB48C36DDD44B416D23654034A66F403BA511C5EFA3</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1250</ExpiryDate>
<Index>243</Index>


```
<ActivationDate />
<Checksum>128EB33128E63E38C9A83A2B1A9349E178F82196</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>A6DA428387A502D7DDFB7A74D3F412BE762627197B25435B7A81716A700157DDD06F
7CC99D6CA28C2470527E2C03616B9C59217357C2674F583B3BA5C7DCF2838692D023E3562420B
4615C439CA97C44DC9A249CFCE7B3BFB22F68228C3AF13329AA4A613CF8DD853502373D62E49A
B256D2BC17120E54AEDCED6D96A4287ACC5C04677DA4A5A320DB8BEE2F775E5FEC5</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1217</ExpiryDate>
  <Index>4</Index>
  <ActivationDate />
  <Checksum>381A035DA58B482EE2AF75F4C3F2CA469BA4AA6C</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>B8048ABC30C90D976336543E3FD7091C8FE4800DF820ED55E7E94813ED005558573F
ECA3D84AF6131A651D66CF4284FB13B635EDD0EE40176D8BF04B7FD1C7BACF9AC7327DFAA8AA
72D10DB3B8E70B2DD0811CB4196525EA386ACC33C0D9D4575916469C4E4F53E8E1C912CC618CB
22DDE7C3568E90022E6BBA770202E4522A2DD623D180E215BD1D1507FE3DC90CA310D27B3EFC
D8F83DE3052CAD1E48938C68D095AAC91B5F37E28BB49EC7ED597</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1221</ExpiryDate>
  <Index>5</Index>
  <ActivationDate />
  <Checksum>EBFA0D5D06D8CE702DA3EAE890701D45E274C845</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>CB26FC830B43785B2BCE37C81ED334622F9622F4C89AAE641046B2353433883F307F
B7C974162DA72F7A4EC75D9D657336865B8D3023D3D645667625C9A07A6B7A137CF0C64198AE3
8FC238006FB2603F41F4F3BB9DA1347270F2F5D8C606E420958C5F7D50A71DE30142F70DE4688
89B5E3A08695B938A50FC980393A9CBCE44AD2D64F630BB33AD3F5F5FD495D31F37818C1D9407
1342E07F1BEC2194F6035BA5DED3936500EB82DFDA6E8AFB655B1EF3D0D7EBF86B66DD9F29F6B
1D324FE8B26CE38AB2013DD13F611E7A594D675C4432350EA244CC34F3873CBA06592987A1D7E
852ADC22EF5A2EE28132031E48F74037E3B34AB747F</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1221</ExpiryDate>
  <Index>6</Index>
  <ActivationDate />
  <Checksum>F910A1504D5FFB793D94F3B500765E1ABCAD72D9</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
```

```
<CAPublicKeyCertificate>  
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>  
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>
```

```
<Modulus>A0DCFA4BDE19C3546B4B6F0414D174DDE294AABBB828C5A834D73AAE27C99B0B053A9  
0278007239B6459FF0BBBCD7B4B9C6C50AC02CE91368DA1BD21AAEDBC65347337D89B68F5C99A  
09D05BE02DD1F8C5BA20E2F13FB2A27C41D3F85CAD5CF6668E75851EC66EDBF98851FD4E42C44  
C1D59F5984703B27D5B9F21B8FA0D93279FBBF69E090642909C9EA27F898959541AA6757F5F62  
4104F6E1D3A9532F2A6E51515AEAD1B43B3D7835088A2FAFA7BE7</Modulus>
```

```
  <Exponent>03</Exponent>  
  <ExpiryDate>1250</ExpiryDate>  
  <Index>241</Index>  
  <ActivationDate />  
  <Checksum>D8E68DA167AB5A85D8C3D55ECB9B0517A1A5B4BB</Checksum>  
  <RID>A000000004</RID>
```

```
</CAPublicKeyCertificate>
```

```
<CAPublicKeyCertificate>
```

```
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>  
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>
```

```
<Modulus>98F0C770F23864C2E766DF02D1E833DFF4FFE92D696E1642F0A88C5694C6479D16DB  
1537BFE29E4FDC6E6E8AFD180EB7EA0124723C333179BF19E93F1065882F776E829E87DAEDA9C  
94A8B3382199A350C077977C97AFF08FD11310AC950A72C3CA5002EF513FCCC286E646E3C5387  
535D509514B3B326E1234F9C848C36DD44B416D23654034A66F403BA511C5EFA3</Modulus>
```

```
  <Exponent>03</Exponent>  
  <ExpiryDate>1250</ExpiryDate>  
  <Index>243</Index>  
  <ActivationDate />  
  <Checksum>A69AC7603DAF566E972DEDC2CB433E07E8B01A9A</Checksum>  
  <RID>A000000004</RID>
```

```
</CAPublicKeyCertificate>
```

```
<CAPublicKeyCertificate>
```

```
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>  
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>
```

```
<Modulus>A6E6FB72179506F860CCCA8C27F99CECD94C7D4F3191D303BBEE37481C7AA15F233B  
A755E9E4376345A9A67E7994BDC1C680BB3522D8C93EB0CCC91AD31AD450DA30D337662D19AC0  
3E2B4EF5F6EC1828D491E19767D7B24542DFDEFF6F62185503532069BBB369E3BB9F819AC6F1  
C30B97D249EEE764E0BAC97F25C873D973953E5153A42064BBFABFD06A4BB486860BF6637406C  
9FC36813A4A75F75C31CCA9F69F8DE59ADECEFF6BDE7E07800FCBE035D3176AF8473E23E9AA3DF  
EE221196D1148302677C720CFE2544A03DB553E7F1B8427BA1CC72B0F29B12DFEF4C081D076D3  
53E71880AADFF386352AF0AB7B28ED49E1E672D11F9</Modulus>
```

```
  <Exponent>03</Exponent>  
  <ExpiryDate>1250</ExpiryDate>  
  <Index>245</Index>  
  <ActivationDate />  
  <Checksum>C2239804C8098170BE52D6D5D4159E81CE8466BF</Checksum>  
  <RID>A000000004</RID>
```

```
</CAPublicKeyCertificate>
```

```
</ArrayOfCAPublicKeyCertificate>
```

Revoked CA Public Key Certificates:

```
'DCEMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.
[2018/05/16 15:10:01.282] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfRevokedCAPublicKeyCertificate
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    </ArrayOfRevokedCAPublicKeyCertificate>
```

[2018/05/16 15:10:01.299] [CardExceptionHandler]
Terminal Exception File:

```
'DCEMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.
[2018/05/16 15:10:01.474] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfHotCard xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    </ArrayOfHotCard>
```

[2018/05/16 15:10:01.718] [EMVContactTerminalApplication]
TransactionRequest
Amount :3000
AmountOther :0
TransactionTypeEnum :PurchaseGoodsAndServices
TransactionDate :2018/05/16 1:10:00 PM

[2018/05/16 15:10:01.735]
[TerminalSupportedKernelAidTransactionTypeCombinations]
Contact Terminal Supported AIDs:

```
'DCEMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.
[2018/05/16 15:10:01.847] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfTerminalSupportedContactKernelAidTransactionTypeCombination
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <TerminalSupportedContactKernelAidTransactionTypeCombination>
      <AIDEnum>A00000000046000</AIDEnum>

    <ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
      </TerminalSupportedContactKernelAidTransactionTypeCombination>
    <TerminalSupportedContactKernelAidTransactionTypeCombination>
      <AIDEnum>A00000000031010</AIDEnum>

    <ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
      </TerminalSupportedContactKernelAidTransactionTypeCombination>
```

[illegible]

```
<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactKernelAidTransactionTypeCombination>
</ArrayOfTerminalSupportedContactKernelAidTransactionTypeCombination>
```

[2018/05/16 15:10:01.875] [PublicKeyCertificateManager]
CA Public Key Certificates:

[2018/05/16 15:10:01.900] [EMVxCtl]
SetStatusLabel: Contact - PresentCard

```
[2018/05/16 15:10:01.927] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfCAPublicKeyCertificate
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <CAPublicKeyCertificate>
      <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
      <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>A90FCD55AA2D5D9963E35ED0F440177699832F49C6BAB15CDAE5794BE93F934D4462
D5D12762E48C38BA83D8445DEAA74195A301A102B2F114EADA0D180EE5E7A5C73E0C4E11F67A4
3DDAB5D55683B1474CC0627F44B8D3088A492FFAADAD4F42422D0E7013536C3C49AD3D0FAE964
59B0F6B1B6056538A3D6D44640F94467B108867DEC40FAAECDD740C00E2B7A8852D</Modulus>
      <Exponent>03</Exponent>
      <ExpiryDate>/>
      <Index>250</Index>
      <ActivationDate />
      <Checksum />
      <RID>A000000004</RID>
    </CAPublicKeyCertificate>
    <CAPublicKeyCertificate>
      <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
      <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>CBCBDB21CCE3A3369EC6FFAB4C26D8F8660FBC75DBA42115F9EDB4E3D654AAF2434C
CEB57837449D0A0A59683D3A199F2FC36149951EBAA192D1336A043016BE72F443B1A4B636CF7
A464F59E480B79C0A8A68684822278B0922C4800931DEC8C73F58B2DE8B2EBA4203AD62191473
8AB49EF5653A6873AC4216E8127D275DB9E4A9A848CDF85B5BD4719F19A33EF953</Modulus>
      <Exponent>03</Exponent>
      <ExpiryDate>1220</ExpiryDate>
      <Index>173</Index>
      <ActivationDate />
      <Checksum />
      <RID>A000000004</RID>
    </CAPublicKeyCertificate>
    <CAPublicKeyCertificate>
      <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
      <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>A89F25A56FA6DA258C8A8B40427D927B4A1EB4D7EA326BBB12F97DED70AE5E4480F
C9C5E8A972177110A1CC318D06D2F8F5C4844AC5FA79A4DC470BB11ED635699C17081B90F1B98
```

4F12E92C1C529276D8AF8EC7F28492097D8CD5BECEA16FE4088F6CFAB4A1B42328A1B996F9278
B0B7E3311CA5EF856C2F888474B83612A82E4E0D0CD4069A6783140433D50725F</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1217</ExpiryDate>
<Index>7</Index>
<ActivationDate />
<Checksum>B4BC56CC4E88324932CBC643D6898F6FE593B172</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>16</PublicKeyAlgorithmIndicator>

<Modulus>D9FD6ED75D51D0E30664BD157023EAA1FFA871E4DA65672B863D255E81E137A51DE4
F72BCC9E44ACE12127F87E263D3AF9DD9CF35CA4A7B01E907000BA85D24954C2FCA3074825DDD
4C0C8F186CB020F683E02F2DEAD3969133F06F7845166ACEB57CA0FC2603445469811D293BFEF
BAFAB57631B3DD91E796BF850A25012F1AE38F05AA5C4D6D03B1DC2E568612785938BBC9B3CD3
A910C1DA55A5A9218ACE9F7A21287752682F15832A678D6E1ED0B</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1224</ExpiryDate>
<Index>8</Index>
<ActivationDate />
<Checksum>20D213126955DE205ADC2FD2822BD22DE21CF9A8</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>9D912248DE0A4E39C1A7DDE3F6D2588992C1A4095AFBD1824D1BA74847F2BC4926D2
EFD904B4B54954CD189A54C5D1179654F8F9B0D2AB5F0357EB642FEDA95D3912C6576945FAB89
7E7062CAA44A4AA06B8FE6E3DBA18AF6AE3738E30429EE9BE03427C9D64F695FA8CAB4BFE3768
53EA34AD1D76BFCAD15908C077FFE6DC5521ECE5D278A96E26F57359FFAEDA19434B937F1AD9
99DC54C1EB11935B44C18100E857F431A4A5A6BB65114F174C2D7B59FDF237D6BB1DD0916E644
D709DED56481477C75D95CDD68254615F7740EC07F330AC5D67BCD75BF23D28A140826C026BBD
E971A37CD3EF9B8DF644AC385010501EFC6509D7A41</Modulus>

<Exponent>03</Exponent>
<ExpiryDate>1225</ExpiryDate>
<Index>9</Index>
<ActivationDate />
<Checksum>1FF80A40173F52D7D27E0F26A146A1C8CCB29046</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
<HashAlgorithmIndicator>1</HashAlgorithmIndicator>
<PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>996AF56F569187D09293C14810450ED8EE3357397B18A2458EFAA92DA3B6DF6514EC
060195318FD43BE9B8F0CC669E3F844057CBDDF8BDA191BB64473BC8DC9A730DB8F6B4EDE3924
186FFD9B8C7735789C23A36BA0B8AF65372EB57EA5D89E7D14E9C7B6B557460F10885DA16AC92
3F15AF3758F0F03EBD3C5C2C949CBA306DB44E6A2C076C5F67E281D7EF56785DC4D75945E491F
01918800A9E2DC66F60080566CE0DAF8D17EAD46AD8E30A247C9F</Modulus>

<Exponent>03</Exponent>

```
<ExpiryDate>1250</ExpiryDate>
<Index>146</Index>
<ActivationDate />
<Checksum>429C954A3859CEF91295F663C963E582ED6EB253</Checksum>
<RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>ACD2B12302EE644F3F835ABD1FC7A6F62CCE48FFEC622AA8EF062BEF6FB8BA8BC68B
BF6AB5870EED579BC3973E121303D34841A796D6DCBC41DBF9E52C4609795C0CCF7EE86FA1D5C
B041071ED2C51D2202F63F1156C58A92D38BC60BDF424E1776E2BC9648078A03B36FB554375FC
53D57C73F5160EA59F3AFC5398EC7B67758D65C9BFF7828B6B82D4BE124A416AB7301914311EA
462C19F771F31B3B57336000DFF732D3B83DE07052D730354D297BEC72871DCCF0E193F171ABA
27EE464C6A97690943D59BDABB2A27EB71CEEBDFA1176046478FD62FEC452D5CA393296530AA
3F41927ADF434A2DF2AE3054F8840657A26E0FC617</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1250</ExpiryDate>
  <Index>148</Index>
  <ActivationDate />
  <Checksum>C443C43CCF87327D136B804160E47D43B60E6E0F</Checksum>
  <RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>BE9E1FA5E9A803852999C4AB432DB28600DCD9DAB76DFAAA47355A0FE37B1508AC6B
F38860D3C6C2E5B12A3CAAF2A7005A7241EBAA7771112C74CF9A0634652FBCA0E5980C54A6476
1EA101A114E0F0B5572ADD57D010B7C9C887E104CA4EE1272DA66D997B9A90B5A6D624AB6C57E
73C8F919000EB5F684898EF8C3DBEFB330C62660BED88EA78E909AFF05F6DA627B</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1250</ExpiryDate>
  <Index>149</Index>
  <ActivationDate />
  <Checksum>EE1511CEC71020A9B90443B37B1D5F6E703030F6</Checksum>
  <RID>A000000003</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>98F0C770F23864C2E766DF02D1E833DFF4FFE92D696E1642F0A88C5694C6479D16DB
1537BFE29E4FDC6E6E8AFD180EB7EA0124723C333179BF19E93F10658B2F776E829E87DAEDA9C
94A8B3382199A350C077977C97AFF08FD11310AC950A72C3CA5002EF513FCCC286E646E3C5387
535D509514B3B326E1234F9CB48C36DD44B416D23654034A66F403BA511C5EFA3</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1250</ExpiryDate>
  <Index>243</Index>
  <ActivationDate />
  <Checksum>128EB33128E63E38C9A83A2B1A9349E178F82196</Checksum>
  <RID>A000000003</RID>
```

```
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>A6DA428387A502D7DDFB7A74D3F412BE762627197B25435B7A81716A700157D0D06F
7CC99D6CA28C2470527E2C03616B9C59217357C2674F583B3BA5C7DC2838692D023E3562420B
4615C439CA97C44DC9A249CFCE7B3BF822F68228C3AF13329AA4A613CF8DD853502373D62E49A
B256D2BC17120E54AEDCED6D96A4287ACC5C04677D4A5A320DB8BEE2F775E5FEC5</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1217</ExpiryDate>
  <Index>4</Index>
  <ActivationDate />
  <Checksum>381A035DA58B482EE2AF75F4C3F2CA469BA4AA6C</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>B8048ABC30C90D976336543E3FD7091C8FE4800DF820ED55E7E94813ED00555B573F
ECA3D84AF6131A651D66CF4284FB13B635EDD0EE40176D8BF04B7FD1C7BACF9AC7327DFAA8AA
72D10DB3B8E70B2DD811CB4196525EA386ACC33C0D9D4575916469C4E4F53E8E1C912CC618CB
22DDE7C3568E90022E6BBA770202E4522A2DD623D180E215BD1D1507FE3DC90CA310D27B3ECC
D8F83DE3052CAD1E48938C68D095AAC91B5F37E28BB49EC7ED597</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1221</ExpiryDate>
  <Index>5</Index>
  <ActivationDate />
  <Checksum>EBFA0D5D06D8CE702DA3EAE890701D45E274C845</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>CB26FC830B43785B2BCE37C81ED334622F9622F4C89AAE641046B2353433883F307F
B7C974162DA72F7A4EC75D9D657336865B8D3023D3D645667625C9A07A6B7A137CF0C64198AE3
8FC238006FB2603F41F4F3BB9DA1347270F2F5D8C606E420958C5F7D50A71DE30142F70DE4688
89B5E3A08695B938A50FC980393A9CBCE44AD2D64F630BB33AD3F5F5FD495D31F37818C1D9407
1342E07F1BEC2194F6035BA5DED3936500EB82DFDA6E8AFB655B1EF3D0D7EBF86B66DD9F29F6B
1D324FE8B26CE38AB2013DD13F611E7A594D675C4432350EA244CC34F3873CBA06592987A1D7E
852ADC22EF5A2EE28132031E48F74037E3B34AB747F</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1221</ExpiryDate>
  <Index>6</Index>
  <ActivationDate />
  <Checksum>F910A1504D5FFB793D94F3B500765E1ABCAD72D9</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>
```



```

<Modulus>A0DCF48DE19C3546B4B6F0414D174DDE294AABBB828C5A834D73AAE27C99B0B053A9
0278007239B6459FF0BBCD7B4B9C6C50AC02CE91368DA1BD21AAEADBC65347337D89B68F5C99A
09D05BE02DD1F8C58A20E2F13FB2A27C41D3F85CAD5CF6668E75851EC66EDBF98851FD4E42C44
C1D59F5984703B27D5B9F21B8FA0D93279FBBF69E090642909C9EA27F898959541AA6757F5F62
4104F6E1D3A9532F2A6E51515AEAD1B43B3D7835088A2FAFA7BE7</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1250</ExpiryDate>
  <Index>241</Index>
  <ActivationDate />
  <Checksum>D8E68DA167AB5A85D8C3D55ECB9B0517A1A5B4BB</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>98F0C770F23864C2E766DF02D1E833DFF4FFE92D696E1642F0A88C5694C6479D16DB
1537BFE29E4FDC6E6E8AFD1B0EB7EA0124723C333179BF19E93F10658B2F776E829E87DAEDA9C
94A8B3382199A350C077977C97AFF08FD11310AC950A72C3CA5002EF513FCCC286E646E3C5387
535D509514B3B326E1234F9CB48C36DD44B416D23654034A66F403BA511C5EFA3</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1250</ExpiryDate>
  <Index>243</Index>
  <ActivationDate />
  <Checksum>A69AC7603DAF566E972DEDC2CB433E07E8B01A9A</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
<CAPublicKeyCertificate>
  <HashAlgorithmIndicator>1</HashAlgorithmIndicator>
  <PublicKeyAlgorithmIndicator>1</PublicKeyAlgorithmIndicator>

<Modulus>A6E6FB72179506F860CCCA8C27F99CECD94C7D4F3191D303BBEE37481C7AA15F233B
A755E9E4376345A9A67E7994BDC1C680BB3522D8C93EB0CCC91AD31AD450DA30D337662D19AC0
3E2B4EF5F6EC18282D491E19767D7B24542DFDEF6F62185503532069BBB369E3BB9FB19AC6F1
C30B97D249EEE764E0BAC97F25C873D973953E5153A42064BBFABFD06A4BB486860BF6637406C
9FC36813A4A75F75C31CCA9F69F8DE59ADECEF6BDE7E07800FCBE035D3176AF8473E23E9AA3DF
EE221196D1148302677C720CFE2544A03DB553E7F1B8427BA1CC72B0F29B12DDEF4C081D076D3
53E71880AADFF386352AF0AB7B28ED49E1E672D11F9</Modulus>
  <Exponent>03</Exponent>
  <ExpiryDate>1250</ExpiryDate>
  <Index>245</Index>
  <ActivationDate />
  <Checksum>C2239804C8098170BE52D6D5D4159E81CE8466BF</Checksum>
  <RID>A000000004</RID>
</CAPublicKeyCertificate>
</ArrayOfCAPublicKeyCertificate>

```

[2018/05/16 15:10:01.996] [PublicKeyCertificateManager]
 Revoked CA Public Key Certificates:

[2018/05/16 15:10:02.001] [XMLUtil`1]

```
<?xml version = "1.0" encoding="utf-8"?>
    <ArrayOfRevokedCAPublicKeyCertificate
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    </ArrayOfRevokedCAPublicKeyCertificate>
```

[2018/05/16 15:10:02.006] [CardExceptionHandler]
Terminal Exception File:

```
[2018/05/16 15:10:02.020] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
    <ArrayOfHotCard xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    </ArrayOfHotCard>
```

[2018/05/16 15:10:02.094]
[TerminalSupportedKernelAidTransactionTypeCombinations]
Contactless Terminal Supported AIDs:

'DCMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.

```
[2018/05/16 15:10:02.286] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
```

```
<ArrayOfTerminalSupportedContactlessKernelAidTransactionTypeCombination
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
        <KernelEnum>Kernel2</KernelEnum>
        <RIDEnum>A000000004</RIDEnum>
```

```
<TransactionTypeEnum>PurchaseGoodsAndServices</TransactionTypeEnum>
    <StatusCheckSupportFlag xsi:nil="true" />
    <ZeroAmountAllowedFlag xsi:nil="true" />
    <ReaderContactlessTransactionLimit xsi:nil="true" />
    <ReaderContactlessFloorLimit xsi:nil="true" />
    <TerminalFloorLimit_9F1B xsi:nil="true" />
    <ReaderCVMRequiredLimit xsi:nil="true" />
    <ExtendedSelectionSupportFlag xsi:nil="true" />
```

```
<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
    </TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
        <KernelEnum>Kernel2</KernelEnum>
        <RIDEnum>A000000004</RIDEnum>
        <TransactionTypeEnum>PurchaseWithCashback</TransactionTypeEnum>
        <StatusCheckSupportFlag xsi:nil="true" />
        <ZeroAmountAllowedFlag xsi:nil="true" />
        <ReaderContactlessTransactionLimit xsi:nil="true" />
        <ReaderContactlessFloorLimit xsi:nil="true" />
        <TerminalFloorLimit_9F1B xsi:nil="true" />
```

```

        <ReaderCVMRequiredLimit xsi:nil="true" />
        <ExtendedSelectionSupportFlag xsi:nil="true" />

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactlessKernelAidTransactionTypeCombination>
  <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <KernelEnum>Kernel12</KernelEnum>
    <RIDEnum>A000000004</RIDEnum>
    <TransactionTypeEnum>Refund</TransactionTypeEnum>
    <StatusCheckSupportFlag xsi:nil="true" />
    <ZeroAmountAllowedFlag xsi:nil="true" />
    <ReaderContactlessTransactionLimit xsi:nil="true" />
    <ReaderContactlessFloorLimit xsi:nil="true" />
    <TerminalFloorLimit_9F1B xsi:nil="true" />
    <ReaderCVMRequiredLimit xsi:nil="true" />
    <ExtendedSelectionSupportFlag xsi:nil="true" />

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactlessKernelAidTransactionTypeCombination>
  <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <TTQAsBytes>9F660427C04000</TTQAsBytes>
    <KernelEnum>Kernel13</KernelEnum>
    <RIDEnum>A000000050</RIDEnum>

<TransactionTypeEnum>PurchaseGoodsAndServices</TransactionTypeEnum>
  <StatusCheckSupportFlag>false</StatusCheckSupportFlag>
  <ZeroAmountAllowedFlag>true</ZeroAmountAllowedFlag>

<ReaderContactlessTransactionLimit>100000</ReaderContactlessTransactionLimit>
  <ReaderContactlessFloorLimit>30000</ReaderContactlessFloorLimit>
  <TerminalFloorLimit_9F1B>20000</TerminalFloorLimit_9F1B>
  <ReaderCVMRequiredLimit>10000</ReaderCVMRequiredLimit>

<ExtendedSelectionSupportFlag>false</ExtendedSelectionSupportFlag>

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactlessKernelAidTransactionTypeCombination>
  <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <TTQAsBytes>9F660427C04000</TTQAsBytes>
    <KernelEnum>Kernel13</KernelEnum>
    <RIDEnum>A000000003</RIDEnum>

<TransactionTypeEnum>PurchaseGoodsAndServices</TransactionTypeEnum>
  <StatusCheckSupportFlag>false</StatusCheckSupportFlag>
  <ZeroAmountAllowedFlag>true</ZeroAmountAllowedFlag>

<ReaderContactlessTransactionLimit>100000</ReaderContactlessTransactionLimit>
  <ReaderContactlessFloorLimit>30000</ReaderContactlessFloorLimit>
  <TerminalFloorLimit_9F1B>20000</TerminalFloorLimit_9F1B>
  <ReaderCVMRequiredLimit>10000</ReaderCVMRequiredLimit>

<ExtendedSelectionSupportFlag>false</ExtendedSelectionSupportFlag>

```

```

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
      <TTQAsBytes>9F660427C04000</TTQAsBytes>
      <KernelEnum>Kernel13</KernelEnum>
      <RIDEnum>A000000003</RIDEnum>
      <TransactionTypeEnum>PurchaseWithCashback</TransactionTypeEnum>
      <StatusCheckSupportFlag>false</StatusCheckSupportFlag>
      <ZeroAmountAllowedFlag>true</ZeroAmountAllowedFlag>

<ReaderContactlessTransactionLimit>100000</ReaderContactlessTransactionLimit>
  <ReaderContactlessFloorLimit>30000</ReaderContactlessFloorLimit>
  <TerminalFloorLimit_9F1B>20000</TerminalFloorLimit_9F1B>
  <ReaderCVMRequiredLimit>10000</ReaderCVMRequiredLimit>

<ExtendedSelectionSupportFlag>false</ExtendedSelectionSupportFlag>

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactlessKernelAidTransactionTypeCombination>
    <TerminalSupportedContactlessKernelAidTransactionTypeCombination>
      <TTQAsBytes>9F660427C04000</TTQAsBytes>
      <KernelEnum>Kernel13</KernelEnum>
      <RIDEnum>A000000003</RIDEnum>
      <TransactionTypeEnum>Refund</TransactionTypeEnum>
      <StatusCheckSupportFlag>false</StatusCheckSupportFlag>
      <ZeroAmountAllowedFlag>true</ZeroAmountAllowedFlag>

<ReaderContactlessTransactionLimit>100000</ReaderContactlessTransactionLimit>
  <ReaderContactlessFloorLimit>30000</ReaderContactlessFloorLimit>
  <TerminalFloorLimit_9F1B>20000</TerminalFloorLimit_9F1B>
  <ReaderCVMRequiredLimit>10000</ReaderCVMRequiredLimit>

<ExtendedSelectionSupportFlag>false</ExtendedSelectionSupportFlag>

<ApplicationSelectionIndicator>true</ApplicationSelectionIndicator>
  </TerminalSupportedContactlessKernelAidTransactionTypeCombination>

</ArrayOfTerminalSupportedContactlessKernelAidTransactionTypeCombination>

```

[2018/05/16 15:10:06.729] [EMVTxCtl]
 SetStatusLabel: Contactless - PresentCard

Status: SharedATR [19] = 3B-8E-80-01-80-31-80-66-B1-84-0C-01-6E-01-83-00-90-00-1C
 Exception thrown: 'System.Runtime.InteropServices.COMException' in
 DCEMV_WindowsDevicesSmartCardsDriver.dll
 WinRT information: The operation attempted to access data outside the valid
 range

The operation attempted to access data outside the valid range

The operation attempted to access data outside the valid range
at Windows.Storage.Streams.DataReader.ReadBuffer(UInt32 length)
at PcsC.Common.IccDetection.DetectCard()
at PcsC.Common.IccDetection.<DetectCardTypeAsync>d__25.MoveNext()

Connected to card

PC/SC device class: Unknown

Card name: Unknown

ATR: 3B-8E-80-01-80-31-80-66-B1-84-0C-01-6E-01-83-00-90-00-1C

[2018/05/16 15:10:16.953] [EMVCommand]

Start ADPU Request: EMVSelectPPSERequest

Filename: 2PAY.SYS.DDF01[325041592E5359532E4444463031]

End ADPU Request: EMVSelectPPSERequest

[2018/05/16 15:10:17.013] [CardQProcessor]

ApuCommand CLA=00,INS=A4,P1=04,Data=325041592E5359532E4444463031

Raw Data Sent:00A404000E325041592E5359532E444446303100

[2018/05/16 15:10:17.079] [CardQProcessor]

Raw Data

Received:6F21840E325041592E5359532E4444463031A50FBF0C0C610A4F08A0000000500101019000

[2018/05/16 15:10:17.125] [EMVResponse]

Start ADPU Response:EMVSelectPPSEReponse

Status: Command successfully executed (OK).

T:[6F] File Control Information (FCI) Template L:[33]

V:[

T:[84] Dedicated File (DF) Name L:[14]

V:[325041592E5359532E4444463031]

T:[A5] File Control Information (FCI) Proprietary Template

L:[15]

V:[

T:[BF0C] File Control Information (FCI) Issuer

Discretionary Data L:[12] V:[

T:[61] Application Template L:[10]

V:[

T:[4F] Application Dedicated File

(ADF) Name L:[8]

V:[A000000050010101]]]]]

End ADPU Response:EMVSelectPPSEReponse

[2018/05/16 15:10:17.200] [EMVCommand]

Start ADPU Request: EMVSelectApplicationRequest

AID: A000000050010101

End ADPU Request: EMVSelectApplicationRequest

[2018/05/16 15:10:17.246] [CardQProcessor]

ApduCommand CLA=00,INS=A4,P1=04,P2=00,Data=A000000050010101
Raw Data Sent:00A4040008A00000005001010100

[2018/05/16 15:10:17.279] [CardQProcessor]
Raw Data
Received:6F20840754657374417070A5155007546573744170709F38099F02069F37049F6604
9000

[2018/05/16 15:10:17.287] [EMVResponse]
Start ADPU Response:EMVSelectApplicationResponse
Status: Command successfully executed (OK).
T:[6F] File Control Information (FCI) Template L:[32]
V:[
 T:[84] Dedicated File (DF) Name L:[7]
V:[54657374417070]
 T:[A5] File Control Information (FCI) Proprietary Template
L:[21]
 V:[
 T:[50] Application Label L:[7]
V:[TestApp]
 T:[9F38] Processing Options Data Object List (PDOL)
L:[9]
 V:[9F02069F37049F6604]]]
End ADPU Response:EMVSelectApplicationResponse

[2018/05/16 15:10:17.331] [KernelDatabaseBase]
Using Global Kernel 3 Values:

'DCEMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.
[2018/05/16 15:10:17.496] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
 <Kernel3Configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <IDSSupported>false</IDSSupported>
 <FDDAForOnlineSupported>true</FDDAForOnlineSupported>
 <SDAForOnlineSupported>true</SDAForOnlineSupported>

 <DisplayAvailableSpendingAmount>true</DisplayAvailableSpendingAmount>
 <AUCManualCheckSupported>true</AUCManualCheckSupported>
 <AUCCashbackCheckSupported>true</AUCCashbackCheckSupported>
 <ATMOfflineCheck>true</ATMOfflineCheck>
 <ExceptionFileEnabled>true</ExceptionFileEnabled>
 </Kernel3Configuration>

[2018/05/16 15:10:17.518] [KernelDatabaseBase]
Using Kernel 3 Defaults:

'DCEMV_DemoApp.UWP.exe' (CoreCLR: CoreCLR_UWP_Domain): Loaded
'Microsoft.GeneratedCode'.

```
[2018/05/16 15:10:17.786] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfTLVXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <TLVXML>
      <Tag>9F33</Tag>
      <Value>E0F8F8</Value>
    </TLVXML>
    <TLVXML>
      <Tag>9F40</Tag>
      <Value>0000000000</Value>
    </TLVXML>
    <TLVXML>
      <Tag>9F09</Tag>
      <Value>0002</Value>
    </TLVXML>
    <TLVXML>
      <Tag>9F1A</Tag>
      <Value>0710</Value>
    </TLVXML>
    <TLVXML>
      <Tag>9F35</Tag>
      <Value>22</Value>
    </TLVXML>
    <TLVXML>
      <Tag>9C</Tag>
      <Value>00</Value>
    </TLVXML>
  </ArrayOfTLVXML>
```

```
[2018/05/16 15:10:17.815] [KernelDatabaseBase]
Transaction Type: PurchaseGoodsAndServices Using Kernel3 Defaults:
  T:[9F33] Terminal Capabilities L:[3]
V:[E0F8F8]
  T:[9F40] Additional Terminal Capabilities L:[5]
V:[0000000000]
  T:[9F09] Application Version Number Terminal L:[2]
V:[0002]
  T:[9F1A] Terminal Country Code L:[2]
V:[0710]
  T:[9F35] Terminal Type L:[1]
V:[22]
  T:[9C] Transaction Type L:[1]
V:[00]
```

```
[2018/05/16 15:10:17.838] [XMLUtil`1]
<?xml version = "1.0" encoding="utf-8"?>
  <ArrayOfTLVXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <TLVXML>
      <Tag>DF8112</Tag>
      <Value />
    </TLVXML>
  </ArrayOfTLVXML>
```

```
</TLVXML>
<TLVXML>
  <Tag>FF8102</Tag>
  <Children />
</TLVXML>
<TLVXML>
  <Tag>FF8103</Tag>
  <Children />
</TLVXML>
<TLVXML>
  <Tag>DF8110</Tag>
  <Value>01</Value>
</TLVXML>
<TLVXML>
  <Tag>9F7C</Tag>
  <Value></Value>
</TLVXML>
<TLVXML>
  <Tag>9F53</Tag>
  <Value>32</Value>
</TLVXML>
<TLVXML>
  <Tag>9F45</Tag>
  <Value>0000</Value>
</TLVXML>
<TLVXML>
  <Tag>9F4C</Tag>
  <Value>0000</Value>
</TLVXML>
<TLVXML>
  <Tag>9F39</Tag>
  <Value>00</Value>
</TLVXML>
<TLVXML>
  <Tag>9F1A</Tag>
  <Value>0710</Value>
</TLVXML>
<TLVXML>
  <Tag>9F1D</Tag>
  <Value>00</Value>
</TLVXML>
<TLVXML>
  <Tag>9F1B</Tag>
  <Value>2000</Value>
</TLVXML>
<TLVXML>
  <Tag>5F2A</Tag>
  <Value>0710</Value>
</TLVXML>
</ArrayOfTLVXML>
```

[2018/05/16 15:10:17.853] [TerminalConfigurationData]

Using Terminal Defaults:

T:[DF8112] Tags To Read L:[0]
V:[]
T:[FF8102] Tags To Write Before Gen AC L:[0]
V:[]
T:[FF8103] Tags To Write After Gen AC L:[0]
V:[]
T:[DF8110] Proceed To First Write Flag L:[1]
V:[01]
T:[9F7C] Customer Exclusive Data (CED) L:[0]
V:[]
T:[9F53] Transaction Category Code L:[1]
V:[2]
T:[9F45] Data Authentication Code L:[2]
V:[0000]
T:[9F4C] ICC Dynamic Number L:[2]
V:[0000]
T:[9F39] Point-of-Service (POS) Entry Mode L:[1]
V:[00]
T:[9F1A] Terminal Country Code L:[2]
V:[0710]
T:[9F1D] Terminal Risk Management Data L:[1]
V:[00]
T:[9F1B] Terminal Floor Limit L:[2]
V:[2000]
T:[5F2A] Transaction Currency Code L:[2]
V:[0710]

[2018/05/16 15:10:17.884] [StateEngine]
Executing New Action: Execute_Idle From Signal Enum: START

[2018/05/16 15:10:17.943] [CommonRoutines]
Packing tag: T:[9F02] (Amount, Authorised (Numeric)) L:[6] V:[]T:[9F37]
(Unpredictable Number) L:[4] V:[]T:[9F66] (Terminal Transaction Qualifiers
(TTQ)) L:[4] V:[]

[2018/05/16 15:10:17.994] [CommonRoutines]
Adding tag: T:[9F02] (Amount, Authorised (Numeric)) L:[6] V:[000000003000]

[2018/05/16 15:10:18.021] [CommonRoutines]
Adding tag: T:[9F37] (Unpredictable Number) L:[4] V:[43B55C49]

[2018/05/16 15:10:18.032] [CommonRoutines]
Adding tag: T:[9F66] (Terminal Transaction Qualifiers (TTQ)) L:[4]
V:[27004000]

[2018/05/16 15:10:18.044] [EMVCommand]
Start ADPU Request: EMVGetProcessingOptionsRequest
830E00000000300043B55C4927004000
End ADPU Request: EMVGetProcessingOptionsRequest

[2018/05/16 15:10:18.067] [StateEngine]

```
[2018/05/16 15:10:18.072] [CardQProcessor]
*****
*****
ApduCommand CLA=80, INS=A8, P1=00, P2=00, Data=830E00000000300043B55C4927004000
Raw Data Sent: 80A8000010830E00000000300043B55C492700400000
```

```

[2018/05/16 15:10:15.272] [EMVResponse]
Start ADPU Response:EMVGetProcessingOptionsResponse
Status: Command successfully executed (OK).
T:[77] Response Message Template Format 2 L:[157]
V:[
    T:[82] Application Interchange Profile L:[2]
V:[0000]
    T:[57] Track 2 Equivalent Data L:[12]
V:[1234567890123456D2512201]
    T:[5F20] Cardholder Name L:[2]
V:[ /]
    T:[5A] Application Primary Account Number (PAN) L:[8]
V:[1234567890123456]
    T:[5F34] Application Primary Account Number (PAN) Sequence
Number L:[1]
    V:[01]
    T:[9F10] Issuer Application Data L:[32]
V:[00000000A0000000000000000000000000000000000000000000000000000000]
    T:[9F26] Application Cryptogram L:[8]
V:[6BD261F1264A4B4B]
    T:[9F27] Cryptogram Information Data L:[1]
V:[80]
    T:[9F36] Application Transaction Counter (ATC) L:[2]
V:[0002]
    T:[9F4B] Signed Dynamic Application Data L:[15]
V:[00000000000000000000000000000000]
    T:[9F6C] Card Transaction Qualifiers (CTQ) L:[2]
V:[2800]
    T:[9F6E] Form Factor Indicator (FFI) L:[4]
V:[00000000]
    T:[9F7C] Customer Exclusive Data (CED) L:[32]
V:[0000000000000000000000000000000000000000000000000000000000000000]
End ADPU Response:EMVGetProcessingOptionsResponse

```

[2018/05/16 15:10:18.298] [KernelDatabaseBase]
Parsing Result:True

[2018/05/16 15:10:18.355] [StateEngine]
Executing New Action: Execute_EXIT From Signal Enum: STOP

[2018/05/16 15:10:18.364] [EMVTerminalApplicationBase]
Terminal received signal:UI

[2018/05/16 15:10:18.393] [EMVTxCtl]
SetStatusLabel: Contactless - ClearDisplay

[2018/05/16 15:10:18.436] [EMVTerminalApplicationBase]
T:[DF8116] User Interface Request Data L:[22]
V:[

Kernel1MessageidentifierEnum->CLEAR_DISPLAY
Kernel1StatusEnum->CARD_READ_SUCCESSFULLY
HoldTime->000000
ValueQualifierEnum->NONE
LanguagePreference->0000000000000000
ValueQualifier->000000000000
CurrencyCode->0000

]

[2018/05/16 15:10:18.461] [EMVTerminalApplicationBase]
Terminal received signal:OUT

[2018/05/16 15:10:18.474] [EMVTxCtl]
SetStatusLabel: Contactless - Authorizing

[2018/05/16 15:10:18.485] [EMVTerminalApplicationBase]
T:[DF8116] User Interface Request Data L:[22]
V:[

Kernel1MessageidentifierEnum->AUTHORISING_PLEASE_WAIT
Kernel1StatusEnum->NOT_READY
HoldTime->000000
ValueQualifierEnum->NONE
LanguagePreference->0000000000000000
ValueQualifier->000000000000
CurrencyCode->0000

]

[2018/05/16 15:10:18.498] [EMVTerminalApplicationBase]
T:[DF8129] Outcome Parameter Set L:[8]
V:[

Status->ONLINE_REQUEST
Start->N_A
OnlineResponseData->N_A
CVM->NO_CVM
UIRequestOnOutcomePresent->True

[illegible]