Verifone

VIM .Net Programmers' Guide

Produced 02/09/2020

TABLE OF CONTENTS

1 References

Nr.	Title	Version/Date
[1]	Nexo Retailer protocol (previously EPAS Retailer)	
[2]	Builder pattern	

2 Definitions

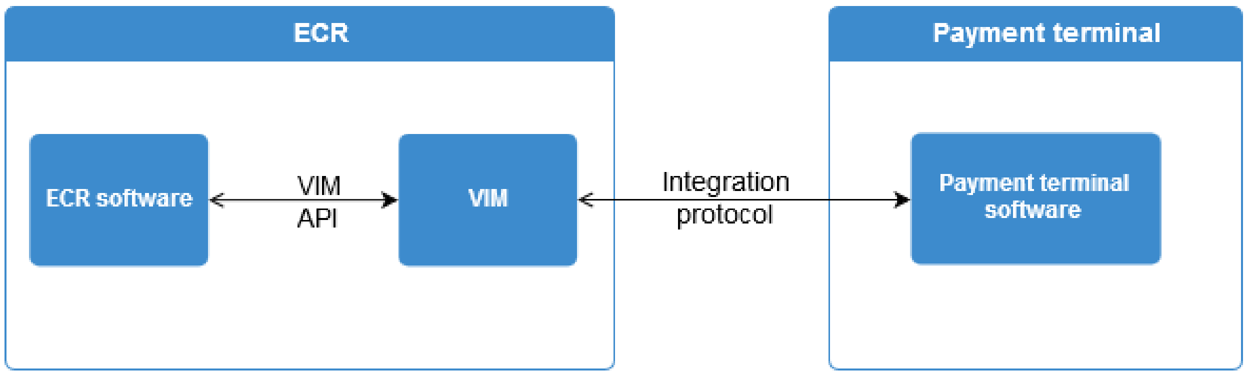
The following terms are used in this document:

Term	Definition
VIM	Point Integration Module
ECR	Electronic Cash Register
POS	Point Of Sale
PA	Payment application
SO	Shared Object
VAS	Value Added Services

3 Introduction

3.1 Purpose

The Verifone Integration Module (VIM) is created to ease the integration of a payment terminal to an ECR application.



The purpose of this document is to describe the VIM programming interface and how to use it correctly between a Verifone payment terminal and the ECR application.

3.2 Updates

Verifone will normally only provide new VIM versions when implementation of new functionality is required. All new functionality will be implemented without breaking backwards compability if possible. The intention is always to keep VIM backwards compatible. If backwards compatibility is not possible in a new version, it will be clearly stated along with the actual changes and VIM or terminal software version dependencies resulting from these changes. If the ECR system does not need to use any of the new functionality in the terminals, an upgrade should not be necessary. However, Verifone strongly recommends that new versions are implemented as soon as possible, as new versions could include general bugfixes and stability improvements etc. Verifone will normally not perform support, bugfixes or improvements on older versions, but first demand that the latest version is being used.

3.3 Scope

This document is intended for developers using VIM to integrate ECR applications with Verifone payment terminals.

4 General description - How to use VIM

VIM has an object oriented ECR interface. The builder pattern (see references) has been used to enable the POS software to easily build the parameter objects required to start operations against VIM. This design helps the ECR application create objects with potentially many parameters and ensure that the objects aren't changed after creation. This is critical to prevent the ECR application from starting an operation, and altering parameters, while VIM is processing the transaction. This pattern was developed with Java in mind, but the structure has been kept for other programming languages to keep consistency.

In broad terms the ECR-application must do the following (see Figure 1: ECR application operation):

- Create a VIM instance providing the needed configuration. See chapter 5.4.1 and 5.1 for more information about configuration of the component. If proper configuration is not provided, an exception/error will be thrown.
- The ECR software needs to set up the following listeners:
    - *vim.SetTerminalConnectListener()* to listen for connection requests from termninals with the possibility to accept or reject the connection (Optional).
    - vim.AddVimListener() to listen for general events and error events.

When a terminal connects, the ECR software will receive a callback providing a terminal instance. The ECR software can then use the *Login* function to establish a session with the terminal. After a successful login, the ECR software is ready to perform transactions on the terminal.

To start a financial or an administrative transaction, use *StartTransaction* or *StartAdmin*. During the transaction processing, the payment terminal will send status events (indicating that status information is available) and finally a result event containing the transaction result. The data received in a callback function is available to the user application as described in chapter 5.3. When it is time to shut down the ECR or change operator, use logout to close the component. Please note that the login and logout functions should not be called for each transaction. Their purpose is for logging in for software or user sessions.

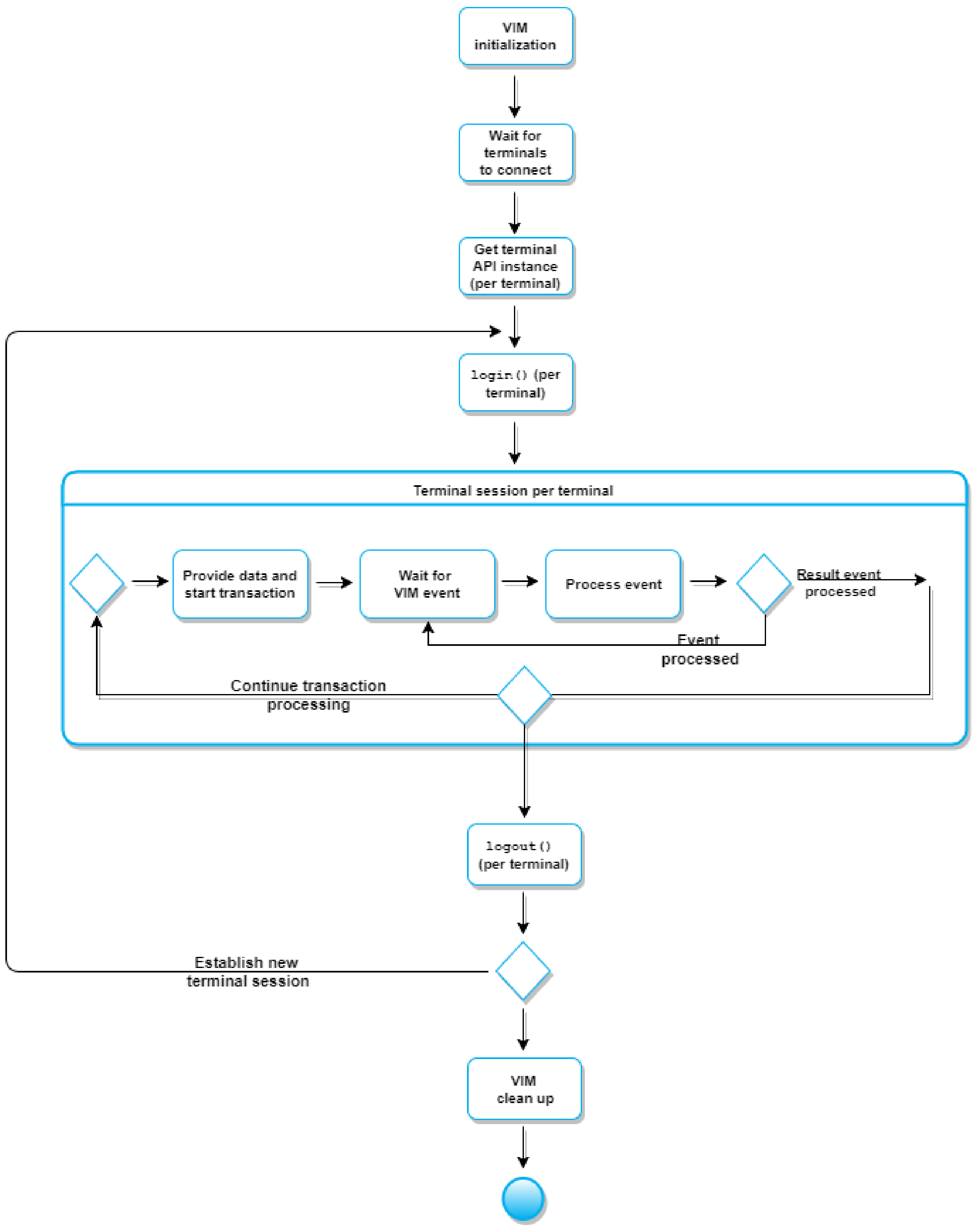

VIM initialization
Wait for terminals to connect
Get terminal API instance (per terminal)
login() (per terminal)
Terminal session per terminal
Provide data and start transaction
Wait for VIM event
Process event
Result event processed
Event processed
Continue transaction processing
logout() (per terminal)
Establish new terminal session
VIM clean up

Figure 1: ECR application operation. The chart is a simple representation of a VIM implementation process. Details are left out to provide a clear picture.

4.1 Core functionality

The following functions are defined as Core functionality that has to be implemented to support a basic integration:

- Purchase
- Reversal
- Reconciliation (handled by host in some setups)
- Error handling

All operations in this list must be handled through result callback methods.
Backwards compatibility on core functionality will not be broken without prior notice.

4.2 Which VIM library works on my OS?

This table is to help you select the correct VIM library to use based on your operating system.

Version	Windows	Linux	Android	iOS	macOS	Latest release
Java	XP/7/8/10	✓	4.0.3 or later		✓	1.0.19
Apple				10.0 or later	10.12 or later	1.0.10
.NET	VIM .net currently supports .NET Standard 1.5 and 2.0, .NET Framework 3.5, 4.0, 4.5 or later and .NET CF 3.5. See https://docs.microsoft.com/en-us/dotnet/standard/net-standard for the platforms supported and the version of the .NET implementation required for each platform.					1.0.9

5 Technical specification with code samples

The sample code is provided for educational purposes and to assist development efforts. The intension is to illustrate how common patterns are performed in VIM. As a result, they usually lack error handling and other features that makes them usable for anything else than simple tests.

5.1 VIM initialisation

```
using System;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Configuration;
using Verifone.Vim.Api.Events;
using Verifone.Vim.Api.Listeners;

namespace VimNetSamples
{
    class VimInitialisationDemo : ITerminalConnectListener, IVimListener, IVimApiListener
    {
        IVim vim;
        IVimApi terminalApi;

        public void InitialiseVim()
        {
            // Create VIM instance
            try
            {
                vim = VimFactory.CreateVim(new VimConfig.Builder()
                    .LogLocation(Directory.GetCurrentDirectory())
                    .LogLevel(VimLogLevel.DEBUG)
                    .DefaultCurrency(CurrencyType.SEK)
                    .Build());

            }
            catch (VimConfigException e)
            {
                // Unable to configure VIM
            }

            // Add optional TerminalConnectListener to accept/reject connecting payment terminals
            vim.SetTerminalConnectListener(this);

            // Add mandatory VimListener
            vim.AddVimListener(this);

            // Initialise VIM and start listening for payment terminals
            try
            {
                vim.Initialise();
            }
            catch (InvalidOperationException e)
            {
                // Initialization failed
            }
        }
        public void OnTerminalConnect(TerminalConnectEvent connectEvent)
        {
            // Get payment terminal information
            // Accept payment terminal connecting to VIM
            connectEvent.AcceptTerminal();
        }

        public void OnTerminalReady(TerminalReadyEvent readyEvent)
        {
            // Get VimApi instance for interacting with the payment terminal
            Terminal terminal = readyEvent.Terminal;
            terminalApi = terminal.GetVimApi(this);
        }

        public void OnConnectionChangedEvent(ConnectionChangedEvent connectionChangedEvent)
        {
            // Handle connection change events
        }

        public void OnCommunicationErrorEvent(CommunicationErrorEvent communicationErrorEvent)
        {
            // Handle communication error events
        }

        public void OnCardEvent(CardEvent cardEvent)
        {
            // Handle card events
        }

        public void OnTerminalMaintenanceEvent(MaintenanceEvent maintenanceEvent)
        {
            // Handle terminal maintenance events
        }

        public void OnBarcodeScanEvent(BarcodeScanEvent barcodeScanEvent)
        {
            // Handle barcode events
        }

        public void OnAccountSearchEvent(AccountSearchEvent accountSearchEvent)
        {
            // Handle account search events
        }


        public void OnVimError(VimErrorEvent evnt)
        {
            // Error handling
        }
    }
}
```

5.2 Terminal session management

Methods for handling the session between an ECR and a payment terminal. Parameters for these methods will be described in more detail under the chapter 5.4.

5.2.1 Login

5.2.1.1 Description

Establishes a session between the ECR software and a specific payment terminal. The environment, capabilities and profile of the two components are exchanged, with the default value of some information that will be used for the next exchanges.

5.2.1.2 Login example

```
using System.Collections.Generic;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class LoginDemo : ILoginResultListener
    {
        IVimApi terminalApi;
        LoginParameters loginParameters;

        public void Login()
        {
            terminalApi.Login(loginParameters, this);
        }

        public void OnSuccess(LoginResult result)
        {
            // Get information about the successful login
            string terminalSoftwareManufacturer = result.TerminalSoftwareManufacturer;
            string terminalSoftwareName = result.TerminalSoftwareName;
            string terminalSoftwareVersion = result.TerminalSoftwareVersion;
            string terminalSerialNumber = result.TerminalSerialNumber;
            List<TerminalCapabilitiesType> terminalCapabilities = result.TerminalCapabilities;
            TerminalGlobalStatusType terminalGlobalStatus = result.TerminalGlobalStatus;
        }

        public void OnFailure(LoginFailureResult result)
        {
            // Get information about the failed login
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;
        }
        public void OnTimeout(TimeoutReason reason)
        {
            // Login timed out
        }
    }
}
```

5.2.1.3 Parameters

LoginParameters: class
ILoginResultListener: interface

5.2.1.4 Callback

void **OnSuccess**(LoginResult result)
void **OnFailure**(LoginFailureResult result)
void **OnTimeout**(TimeoutReason reason)

5.2.2 Logout

5.2.2.1 Description

Ends the session between the ECR software and the payment terminal, which was previously created by a Login.

5.2.2.2 Logout example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
   class LogoutDemo : ILogoutResultListener
   {
      IVimApi terminalApi;
      LogoutParameters logoutParameters;

      public void Logout()
      {
         terminalApi.Logout(logoutParameters, this);
      }

      public void OnSuccess(LogoutResult result)
      {
         // Logout success
      }

      public void OnFailure(LogoutFailureResult result)
      {
         // Logout failed
      }

      public void OnTimeout(TimeoutReason reason)
      {
         // Logout timed out
      }
   }
}
```

5.2.2.3 Parameters

LogoutParameters: class
ILogoutResultListener: interface

5.2.2.4 Callback

void **OnSuccess**(LogoutResult result)
void **OnFailure**(LogoutFailureResult result)
void **OnTimeout**(TimeoutReason reason)

5.3 Transaction processing methods

Methods to start administrative and financial transactions on a initialised terminal instance. Parameters for these methods will be described in more detail under the chapter 5.4.

5.3.1 StartTransaction

5.3.1.1 Description

Executes the purchase, payment of goods or services by the ECR operator/SW towards the payment terminal. Including normal purchase, the transaction type and related parameters can be changed to create other transaction types, such as refund, deferred sale, reservation, etc.

5.3.1.2 StartTransaction example

```
using System;
using System.Collections.Generic;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Content;
using Verifone.Vim.Api.Common.PaymentInstrumentData;
using Verifone.Vim.Api.Common.Receipt;
using Verifone.Vim.Api.Common.Token;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class TransactionDemo : ITransactionResultListener
    {
        IVimApi vimApi;
        TransactionParameters transactionParameters;

        public void StartTransaction()
        {
            vimApi.StartTransaction(transactionParameters, this);
        }

        public void OnSuccess(TransactionResult result)
        {
            // Get information about the approved transaction
            TransactionType transType = result.TransactionType;
            decimal transAmount = result.AuthorizedAmount;
            decimal cashbackAmount = result.CashbackAmount;
            decimal tipAmount = result.TipAmount;
            TransactionId ecrTransId = result.EcrTransactionId;
            TransactionId terminalTransId = result.TerminalTransactionId;
            string approvalCode = result.ApprovalCode;
            PaymentInstrumentData instrumentData = result.PaymentInstrumentData;
            PaymentInstrumentType? instrumentType = instrumentData?.PaymentInstrumentType;
            if (instrumentType == PaymentInstrumentType.Card)
            {
                CardData cardData = instrumentData.CardData;
                EntryModeType entryMode = cardData.EntryMode;
                string paymentBrand = cardData.PaymentBrand;
                string maskedPan = cardData.MaskedPan;
            }
            else if (instrumentType == PaymentInstrumentType.AlternativePayment)
            {
                AlternativePaymentData alternativePaymentData=instrumentData.AlternativePaymentData;
                string alternativePaymentBrand = alternativePaymentData.AlternativePaymentBrand;
                EntryModeType entryMode = alternativePaymentData.EntryMode;
                TransactionId providerTransactionId = alternativePaymentData.ProviderTransactionId;
            }

            List<Receipt> receipts = result.Receipts;
            foreach (Receipt receipt in receipts)
            {
                DocumentType documentType = receipt.DocumentType;
                bool? signatureRequired = receipt.SignatureRequired;
                ReceiptContent receiptContent = receipt.Content;
                if (receiptContent.Format == ReceiptFormatType.Text)
                {
                    ContentText contentText = receiptContent.Text;
                    string plainTextReceipt = contentText.PlainText;
                }
            }

            List<Token> tokens = result.Tokens;
            foreach (Token token in tokens)
            {
                TokenType tokenType = token.Type;
                string tokenValue = token.Value;
                DateTime? tokenExpiry = token.Expiry;
                string tokenSchemeId = token.SchemeId;
                EnrolmentStatusType? enrolmentStatusType = token.EnrolmentStatus;
            }

            // TokenResponses give info about the result of token lookups and
            // provides an alternative way to get the token when the lookup was successful.
            List<TokenResponse> tokenResponses = result.TokenResponses;
            foreach (TokenResponse tokenResponse in tokenResponses)
            {
                TokenRequest tokenRequest = tokenResponse.TokenRequest;
                string tokenSchemeId = tokenRequest.SchemeId;
                if (tokenResponse.Result == TokenResultType.Success)
                {
                    Token token = tokenResponse.Token;
                }
                else
                {
                    TokenFailureErrorType? tokenErrorType = tokenResponse.Error;
                    string tokenErrorDescription = tokenResponse.AdditionalReason;
                }
            }
        }
```

```
public void OnFailure(TransactionFailureResult result)
{
    // Get information about the rejected transaction
    TransactionType transType = result.TransactionType;
    TransactionId ecrTransId = result.EcrTransactionId;
    TransactionId terminalTransId = result.TerminalTransactionId;
    FailureErrorType errorType = result.Error;
    string additionalReason = result.AdditionalReason;
    List<Receipt> receipts = result.Receipts;
    foreach (Receipt receipt in receipts)
    {
        DocumentType documentType = receipt.DocumentType;
        bool? signatureRequired = receipt.SignatureRequired;
        ReceiptContent receiptContent = receipt.Content;
        if (receiptContent.Format == ReceiptFormatType.Text)
        {
            ContentText contentText = receiptContent.Text;
            string plainTextReceipt = contentText.PlainText;
        }
    }

    List<Token> tokens = result.Tokens;
    foreach (Token token in tokens)
    {
        TokenType tokenType = token.Type;
        string tokenValue = token.Value;
        DateTime? tokenExpiry = token.Expiry;
        string tokenSchemeId = token.SchemeId;
        EnrolmentStatusType? enrolmentStatusType = token.EnrolmentStatus;
    }

    // TokenResponses give info about the result of token lookups and
    // provides an alternative way to get the token when the lookup was successful.
    List<TokenResponse> tokenResponses = result.TokenResponses;
    foreach (TokenResponse tokenResponse in tokenResponses)
    {
        TokenRequest tokenRequest = tokenResponse.TokenRequest;
        string tokenSchemeId = tokenRequest.SchemeId;
        if (tokenResponse.Result == TokenResultType.Success)
        {
            Token token = tokenResponse.Token;
        }
        else
        {
            TokenFailureErrorType? tokenErrorType = tokenResponse.Error;
            string tokenErrorDescription = tokenResponse.AdditionalReason;
        }
    }
}
```

```
public void OnDisplayRequest(DisplayRequestData displayRequestData)
{
   // Get information about the display message received from the terminal
   List<DisplayOutput> displayMessages = displayRequestData.DisplayOutput;
   foreach (DisplayOutput displayMessage in displayMessages)
   {
      DeviceType displayMessageType = displayMessage.DeviceType;
      DisplayContent displayMessageContent = displayMessage.Content;
      if (displayMessageContent.Format == DisplayFormatType.Text)
      {
         ContentText contentText = displayMessageContent.Text;
         string plainTextDisplayMessage = contentText.PlainText;
      }
   }
}

public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
{
   // Get information about the request for input from the terminal
   InputRequestType inputType = requestData.InputType;
   DeviceType inputDevice = requestData.DeviceType;
   string inputDefault = requestData.DefaultInputString;
   int? inputTimeout = requestData.TimeoutInSeconds;
   int? inputMinLength = requestData.MinLength;
   int? inputMaxLength = requestData.MaxLength;
   DisplayOutput output = requestData.DisplayOutput;
   DeviceType outputDevice = output.DeviceType;
   DisplayContent outputContent = output.Content;
   if (outputContent.Format == DisplayFormatType.Text)
   {
      ContentText contentText = outputContent.Text;
      string plainTextDisplayMessage = contentText.PlainText;
   }
   // Return user input
   inputReceiver.InputText("1234");  //see chapter "Input requests" for details
}

public void OnInputRequestAborted(InputRequestAbortedData inputRequestAbortedData)
{
   // InputRequest aborted
}

public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
{
   // Get information about the requested print received from the terminal
   DocumentType documentType = requestData.DocumentType;
   PrintContent printContent = requestData.Content;
   if (printContent.Format == PrintFormatType.Text)
   {
      ContentText contentText = printContent.Text;
      string plainTextPrint = contentText.PlainText;
   }
   // Handle print
   // Return response
   printReceiver.PrintSuccess();
}
```

```
        public void OnTimeout(TimeoutReason timeoutReason)
        {
            // Transaction timeout
        }
    }
}
```

5.3.1.3 Parameters

TransactionParameters: class
ITransactionResultListener: interface

5.3.1.4 Callback

void **OnSuccess**(TransactionResult result)
void **OnFailure**(TransactionFailureResult result)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.2 StartAdmin

5.3.2.1 Description

Admin commands are initiated by the ECR software to select and start customised administrative services provided by the payment terminal.

5.3.2.2 StartAdmin example

```
using System.Collections.Generic;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Content;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class AdminDemo : IAdminResultListener
    {
        IVimApi terminalApi;
        AdminParameters adminParameters;

        public void StartAdmin()
        {
            terminalApi.StartAdmin(adminParameters, this);
        }

        public void OnSuccess(AdminResult adminResult)
        {
            // Admin transaction approved
        }

        public void OnFailure(AdminFailureResult result)
        {
            // Get information about the rejected admin transaction
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;
        }

        public void OnDisplayRequest(DisplayRequestData requestData)
        {
            // Display message received from terminal
            // Also see onDisplayRequest in StartTransaction example
        }

        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            // Get information about the request for input received from the terminal
            // Return user input or cancel the request.
            // Also see onInputRequest in StartTransaction example
        }
        public void OnInputRequestAborted(InputRequestAbortedData inputRequestAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see onPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason timeoutReason)
        {
            // Timeout
        }
    }
}
```

5.3.2.3 Parameters

AdminParameters: class
IAdminResultListener: interface

5.3.2.4 Callback

void **OnSuccess**(AdminResult result)
void **OnFailure**(AdminFailureResult result)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.3 StartReconciliation

5.3.3.1 Description

Initiates a reconciliation.

5.3.3.2 StartReconciliation example

```
using System.Collections.Generic;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Content;
using Verifone.Vim.Api.Common.PaymentInstrumentData;
using Verifone.Vim.Api.Common.TransactionTotals;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class ReconciliationDemo : IReconciliationResultListener
    {
        IVimApi terminalApi;
        ReconciliationParameters reconciliationParameters;

        public void StartReconciliation()
        {
            terminalApi.StartReconciliation(reconciliationParameters, this);
        }

        public void OnSuccess(ReconciliationResult result)
        {
            ReconciliationType reconciliationType = result.ReconciliationType;
            List<TransactionTotal> transactionTotals = result.TransactionTotals;
            foreach (TransactionTotal transactionTotal in transactionTotals)
            {
                PaymentInstrumentType instrumentType = transactionTotal.PaymentInstrumentType;
                string hostReconciliationId = transactionTotal.HostReconciliationId;
                string cardBrand = transactionTotal.CardBrand;
                CurrencyType? currencyType = transactionTotal.PaymentCurrency;
                List<PaymentTotal> paymentTotals = transactionTotal.PaymentTotals;
                foreach (PaymentTotal paymentTotal in paymentTotals)
                {
                    TransactionTotalType transactionTotalType = paymentTotal.TransactionTotalType;
                    int transactionCount = paymentTotal.TransactionCount;
                    decimal transactionAmount = paymentTotal.TransactionAmount;
                }
            }
        }
        public void OnFailure(ReconciliationFailureResult result)
        {
            // Reconciliation rejected
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;
        }

        public void OnDisplayRequest(DisplayRequestData requestData)
        {
            // Display message received from terminal
            // Also see onDisplayRequest in StartTransaction example
        }

        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            // Get information about the request for input received from the terminal
            // Return user input or cancel the request.
            // Also see onInputRequest in StartTransaction example
        }

        public void OnInputRequestAborted(InputRequestAbortedData inputRequestAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see onPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason timeoutReason)
        {
            // Timeout
        }
    }
}
```

5.3.3.3 Parameters

ReconciliationParameters: class
IReconciliationResultListener: interface

5.3.3.4 Callback

void **OnSuccess**(ReconciliationResult result)
void **OnFailure**(ReconciliationFailureResult result)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.4 TransactionStatus

5.3.4.1 Description

Request from the ECR software to get the result of the previous transaction, because the response message was not received by the ECR software.

5.3.4.2 TransactionStatus example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
   class TransactionStatusDemo : ITransactionStatusResultListener
   {
      IVimApi terminalApi;
      TransactionStatusParameters transactionStatusParameters;

      public void TransactionStatus()
      {
         terminalApi.TransactionStatus(transactionStatusParameters, this);
      }

      public void OnSuccess(TransactionStatusResult statusResult)
      {
         // Get information about the successful transaction status
         // and the result of the original transaction
         if (statusResult.IsTransactionSuccess)
         {
            TransactionResult successResult = statusResult.TransactionResult;
            // Handle approved transaction result
         }
         else
         {
            TransactionFailureResult failureResult = statusResult.TransactionFailureResult;
            // Handle rejected transaction result
         }
      }

      public void OnFailure(TransactionStatusFailureResult result)
      {
         // Failed to get transaction status or transaction is still ongoing
         // Get information about the failure
         // Transaction still in progress if errorType==FailureErrorType.InProgress
         FailureErrorType errorType = result.Error;
         string additionalReason = result.AdditionalReason;
      }

      public void OnTimeout(TimeoutReason reason)
      {
         // Timeout
      }
   }
}
```

5.3.4.3 Parameters

TransactionStatusParameters: class
ITransactionStatusResultListener: interface

5.3.4.4 Callback

void **OnSuccess**(TransactionStatusResult result)
void **OnFailure**(TransactionStatusFailureResult result)
void **OnTimeout**(TimeoutReason reason)

5.3.5 Abort

5.3.5.1 Description

Request from the ECR software to abort a current transaction. Even if abort is called the ECR software must still keep waiting for the transaction result.

5.3.5.2 Abort example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class AbortDemo
    {
        IVimApi terminalApi;
        AbortParameters abortParameters;

        public AbortDemo()
        {
            terminalApi.Abort(abortParameters);
        }
    }
}
```

5.3.5.3 Parameters

AbortParameters: class

5.3.5.4 Callback

none

5.3.6 StartInput

5.3.6.1 Description

Request information from the payment terminal. At the moment only barcode scanning is supported.

5.3.6.2 StartInput example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class InputDemo : IInputResultListener
    {
        IVimApi terminalApi;
        InputParameters inputParameters;

        public void StartInput()
        {
            terminalApi.StartInput(inputParameters, this);
        }

        public void OnSuccess(InputResult result)
        {
            // Get information about the successful input request
            DeviceType inputDevice = result.Device;
            Input input = result.Input;
            InputType inputType = input.InputType;
            if (inputType == InputType.Barcode)
            {
                Barcode barcode = input.Barcode;
                BarcodeType barcodeType = barcode.Type;
                string barcodeValue = barcode.Value;
            }
        }

        public void OnFailure(InputFailureResult result)
        {
            // Get information about the rejected input request
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;
        }

        public void OnTimeout(TimeoutReason reason)
        {
            // Timeout
        }
    }
}
```

5.3.6.3 Parameters

InputParameters: class
IInputResultListener: interface

5.3.6.4 Callback

void **OnSuccess**(InputResult inputResult)
void **OnFailure**(InputFailureResult result)
void **OnTimeout**(TimeoutReason reason)

5.3.7 GetCardInfo

5.3.7.1 *Description*

Get information about a card read by the terminal. Can be called after receiving a card inserted event to get information about a card already inserted or in idle to prompt the user of the terminal to insert a card.

5.3.7.2 *GetCardInfo example*

```
using System;
using System.Collections.Generic;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Content;
using Verifone.Vim.Api.Common.Loyalty;
using Verifone.Vim.Api.Common.PaymentInstrumentData;
using Verifone.Vim.Api.Common.SensitiveCardData;
using Verifone.Vim.Api.Common.Token;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
   class CardInfoDemo : ICardInfoResultListener
   {
      IVimApi terminalApi;
      CardInfoParameters cardInfoParameters;

      public void GetCardInfo()
      {
         terminalApi.GetCardInfo(cardInfoParameters, this);
      }

      public void OnSuccess(CardInfoResult result)
      {
         // Get information about the successful card info result
         string ecrId = result.EcrId;
         string terminalId = result.TerminalId;
         TransactionId ecrTransactionId = result.EcrTransactionId;
         TransactionId terminalTransactionId = result.TerminalTransactionId;
         PaymentInstrumentData instrumentData = result.PaymentInstrumentData;
         PaymentInstrumentType? instrumentType = instrumentData?.PaymentInstrumentType;
         if (instrumentType == PaymentInstrumentType.Card)
         {
            CardData cardData = instrumentData.CardData;
            EntryModeType entryMode = cardData.EntryMode;
            string paymentBrand = cardData.PaymentBrand;
            string maskedPan = cardData.MaskedPan;
            SensitiveCardData sensitiveCardData = cardData.SensitiveCardData;
            if (sensitiveCardData != null)
            {
               string pan = sensitiveCardData.Pan;
               List<TrackData> trackData = sensitiveCardData.TrackData;
            }
```

```
        }
        else if (instrumentType == PaymentInstrumentType.AlternativePayment)
        {
            AlternativePaymentData alternativePaymentData=instrumentData.AlternativePaymentData;
            string alternativePaymentBrand = alternativePaymentData.AlternativePaymentBrand;
            EntryModeType entryMode = alternativePaymentData.EntryMode;
            TransactionId providerTransactionId = alternativePaymentData.ProviderTransactionId;
        }

        List<LoyaltyAccount> loyaltyAccounts = result.LoyaltyAccounts;
        foreach (LoyaltyAccount account in loyaltyAccounts)
        {
            string loyaltyBrand = account.LoyaltyBrand;
            LoyaltyAccountId accountId = account.LoyaltyAccountId;
            string loyaltyId = accountId.LoyaltyId;
            EntryModeType? entryMode = accountId.EntryModeType;
            IdentificationType identificationType = accountId.IdentificationType;
            IdentificationSupportType? identificationSupport = accountId.IdentificationSupportType;
        }

        List<Token> tokens = result.Tokens;
        foreach (Token token in tokens)
        {
            TokenType tokenType = token.Type;
            string tokenValue = token.Value;
            DateTime? tokenExpiry = token.Expiry;
            string tokenSchemeId = token.SchemeId;
            EnrolmentStatusType? enrolmentStatusType = token.EnrolmentStatus;
        }

        // TokenResponses give info about the result of token lookups and
        // provides an alternative way to get the token when the lookup was successful.
        List<TokenResponse> tokenResponses = result.TokenResponses;
        foreach (TokenResponse tokenResponse in tokenResponses)
        {
            TokenRequest tokenRequest = tokenResponse.TokenRequest;
            string tokenSchemeId = tokenRequest.SchemeId;
            if (tokenResponse.Result == TokenResultType.Success)
            {
                Token token = tokenResponse.Token;
            }
            else
            {
                TokenFailureErrorType? tokenErrorType = tokenResponse.Error;
                string tokenErrorDescription = tokenResponse.AdditionalReason;
            }
        }
    }

    public void OnFailure(CardInfoFailureResult result)
    {
        // Get information about the rejected card info request
        FailureErrorType errorType = result.Error;
        string additionalReason = result.AdditionalReason;
        TransactionId ecrTransactionId = result.EcrTransactionId;
        TransactionId terminalTransactionId = result.TerminalTransactionId;
    }

    public void OnDisplayRequest(DisplayRequestData requestData)
    {
```

```
    // Display message received from terminal
    // Also see onDisplayRequest in StartTransaction example
}
```

```
        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            // Get information about the request for input received from the terminal
            // Return user input or cancel the request.
            // Also see onInputRequest in StartTransaction example
        }

        public void OnInputRequestAborted(InputRequestAbortedData inputRequestAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see onPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason timeoutReason)
        {
            // Timeout
        }
    }
}
```

5.3.7.3 Parameters

CardInfoParameters: class
ICardInfoResultListener: interface

5.3.7.4 Callback

void **OnSuccess**(CardInfoResult cardInfoResult)
void **OnFailure**(CardInfoFailureResult cardInfoFailureResult)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.8 ResetContext

5.3.8.1 Description

Resets the terminal transaction context. Used for example to ask a customer to remove an inserted card.

5.3.8.2 ResetContext example

```
using System.Collections.Generic;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Content;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
   class ResetContextDemo : IResetContextResultListener
   {
      IVimApi terminalApi;
      ResetContextParameters resetContextParameters;

      public void ResetContext()
      {
         terminalApi.ResetContext(resetContextParameters, this);
      }

      public void OnSuccess(ResetContextResult result)
      {
         // Get information about the successful reset of the terminal context
         string ecrId = result.EcrId;
         string terminalId = result.TerminalId;
         string serviceId = result.ServiceId;
      }

      public void OnFailure(ResetContextFailureResult result)
      {
         // Get information about the rejected reset of the terminal context
         FailureErrorType errorType = result.Error;
         string additionalReason = result.AdditionalReason;
      }

      public void OnDisplayRequest(DisplayRequestData requestData)
      {
         // Display message received from terminal
         // Also see onDisplayRequest in StartTransaction example
      }

      public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
      {
         // Get information about the request for input received from the terminal
         // Return user input or cancel the request.
         // Also see onInputRequest in StartTransaction example
      }
```

```
        public void OnInputRequestAborted(InputRequestAbortedData inputRequestAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see onPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason timeoutReason)
        {
            // Timeout
        }
    }
}
```

5.3.8.3 Parameters

ResetContextParameters: class
IResetContextResultListener: interface

5.3.8.4 Callback

void **OnSuccess**(ResetContextResult resetContextResult)
void **OnFailure**(ResetContextFailureResult failureResult)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.9 StartTerminalPrint

5.3.9.1 Description

Print text and barcodes on payment terminal printer. The payment terminal must be configured to accept print requests from the ECR in order for this functionality to work.

5.3.9.2 StartTerminalPrint example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
   class StartTerminalPrintDemo : IPrintResultListener
   {
      IVimApi terminalApi;
      PrintParameters printParameters;

      public void StartTerminalPrint()
      {
         terminalApi.StartTerminalPrint(printParameters, this);
      }

      public void OnSuccess(PrintResult result)
      {
         // Get information about the successful terminal print
      }

      public void OnFailure(PrintFailureResult result)
      {
         // Get information about the failed terminal print
         FailureErrorType errorType = result.Error;
         string additionalReason = result.AdditionalReason;
      }

      public void OnTimeout(TimeoutReason timeoutReason)
      {
         // Terminal print timeout

      }
   }
}
```

5.3.9.3 Parameters

PrintParameters: class
IPrintResultListener: interface

5.3.9.4 Callback

void **OnSuccess**(PrintResult result)
void **OnFailure**(PrintFailureResult result)
void **OnTimeout**(TimeoutReason reason)

5.3.10 StartBalanceInquiry

5.3.10.1 Description

Balance inquiry is initiated by the ECR software to get the balance of an account, identified by a card (payment card, loyalty card or stored value card) or other identifier.

5.3.10.2 StartBalanceInquiry example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.BalanceInquiry;
using Verifone.Vim.Api.Common.PaymentInstrumentData;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class BalanceInquiryDemo : IBalanceInquiryResultListener
    {
        IVimApi vimApi;
        BalanceInquiryParameters balanceInquiryParameters;

        public void StartTransaction()
        {
            vimApi.StartBalanceInquiry(balanceInquiryParameters, this);
        }

        public void OnSuccess(BalanceInquiryResult result)
        {
            // Get information about the successful balance inquiry result
            string ecrId = result.EcrId;
            string terminalId = result.TerminalId;

            PaymentAccountStatus paymentAccountStatus = result.PaymentAccountStatus;
            PaymentInstrumentData instrumentData = paymentAccountStatus.PaymentInstrumentData;
            PaymentInstrumentType? instrumentType = instrumentData.PaymentInstrumentType;
            // Get information about the payment instrument
            // Also see onSuccess in StartTransaction example

            decimal currentBalance = paymentAccountStatus.CurrentBalance;
            CurrencyType currencyType = paymentAccountStatus.CurrencyType;
        }

        public void OnFailure(BalanceInquiryFailureResult result)
        {
            // Get information about the rejected balance inquiry request
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;
        }

        public void OnDisplayRequest(DisplayRequestData displayRequestData)
        {
            // Display message received from terminal
            // Also see onDisplayRequest in StartTransaction example
        }
        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            // Get information about the request for input received from the terminal
            // Return user input or cancel the request.
            // Also see onInputRequest in StartTransaction example
        }

        public void OnInputRequestAborted(InputRequestAbortedData inputRequestAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see onPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason timeoutReason)
        {
            // Timeout
        }
    }
}
```

5.3.10.3 Parameters

BalanceInquiryParameters: class
IBalanceInquiryResultListener: interface

5.3.10.4 Callback

void **OnSuccess**(BalanceInquiryResult result)
void **OnFailure**(BalanceInquiryFailureResult result)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.11 StartAccountSelection

5.3.11.1 Description

Starts an account selection process on a terminal in a pay at table system. Will typically be used following the reception of an OnAccountSearchEvent() callback. See chapter 5.8.1 for more information about the pay at table use case.

5.3.11.2 StartAccountSelection example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Account;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class StartAccountSelectionDemo : IAccountSelectionResultListener
    {
        IVimApi vimApi;
        AccountSelectionParameters accountSelectionParameters;

        public void StartAccountSelection()
        {
            vimApi.StartAccountSelection(accountSelectionParameters, this);
        }

        public void OnSuccess(AccountSelectionResult result)
        {
            // Account selection completed successfully
            AccountId selectedAccountId = result.AccountId;

            // Allow the terminal to request an account operation to be started.
            // Use StartAccountOperationSelection()
        }

        public void OnFailure(AccountSelectionFailureResult result)
        {
            // Account selection rejected
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;

            // Return to idle
        }

        public void OnDisplayRequest(DisplayRequestData requestData)
        {
            // Display message received from terminal
            // Also see OnDisplayRequest in StartTransaction example
        }

        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            // Get information about the request for input received from the terminal
            // Return user input or cancel the request.
            // Also see OnInputRequest in StartTransaction example
        }

        public void OnInputRequestAborted(InputRequestAbortedData inputAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see OnPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason reason)
        {
            // Timeout
        }
    }
}
```

5.3.11.3 Parameters

AccountSelectionParameters: class
IAccountSelectionResultListener: interface

5.3.11.4 Callback

void **OnSuccess**(AccountSelectionResult result)
void **OnFailure**(AccountSelectionFailureResult result)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.3.12 StartAccountOperationSelection

5.3.12.1 Description

Starts an account operation selection process on a terminal in a pay at table system. See chapter 5.8.1 for more information about the pay at table use case.

5.3.12.2 StartAccountOperationSelection example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Account;
using Verifone.Vim.Api.DeviceRequests.Display;
using Verifone.Vim.Api.DeviceRequests.Input;
using Verifone.Vim.Api.DeviceRequests.Print;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class StartAccountOperationSelectionDemo : IAccountOperationSelectionResultListener
    {
        IVimApi vimApi;
        AccountOperationSelectionParameters accountOperationSelectionParameters;


        public void StartAccountOperationSelection()
        {
            vimApi.StartAccountOperationSelection(accountOperationSelectionParameters, this);
        }

        public void OnSuccess(AccountOperationSelectionResult result)
        {
            // Account operation selection completed successfully
            AccountId accountId = result.AccountId;
            AccountOperationType operation = result.SelectedAccountOperation;

            if (operation == AccountOperationType.CompletePayment)
            {
                // Use StartTransaction() to start a payment for the remaining amount on the account
            }
            else if (operation == AccountOperationType.PartialPayment)
            {
                // Get details about the partial payment
                PartialPaymentDetails paymentDetails = result.PartialPaymentDetails;
                decimal requestedAmount = paymentDetails.RequestedAmount;
                // List<SaleItems> saleItems = paymentDetails.SaleItems;
                // Use StartTransaction() to start a payment
                // for a part of the remaining amount on the account
            }
            else if (operation == AccountOperationType.PrintAccount)
            {
                // Use StartPrint() to print a summary of the account details.
            }
            else if (operation == AccountOperationType.PrintReceipt)
            {
                // Get details about the receipt to be printed
                PrintReceiptDetails printDetails = result.PrintReceiptDetails;
                TransactionId ecrTransactionID = printDetails.EcrTransactionId;
                TransactionId terminalTransactionID = printDetails.TerminalTransactionId;
                // Use StartPrint() to print the receipt for the transaction identified by the transaction IDs.
            }
        }

        public void OnFailure(AccountOperationSelectionFailureResult result)
        {
            // Account operation selection rejected
            FailureErrorType errorType = result.Error;
            string additionalReason = result.AdditionalReason;
            AccountId accountId = result.AccountId;


            if (errorType == FailureErrorType.Cancel)
            {
                // Unlock account and return to idle
            }
            else
            {
               // Allow the terminal to request a new account operation to be started.
                // Use StartAccountOperationSelection()
            }
        }

        public void OnDisplayRequest(DisplayRequestData requestData)
        {
            // Display message received from terminal
            // Also see OnDisplayRequest in StartTransaction example
        }

        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            // Get information about the request for input received from the terminal
            // Return user input or cancel the request.
            // Also see OnInputRequest in StartTransaction example
        }

        public void OnInputRequestAborted(InputRequestAbortedData inputAbortedData)
        {
            // InputRequest aborted
        }

        public void OnPrintRequest(PrintRequestData requestData, IPrintReceiver printReceiver)
        {
            // Get information about the request for print received from the terminal
            // Handle print
            // Return response
            // Also see OnPrintRequest in StartTransaction example
        }

        public void OnTimeout(TimeoutReason reason)
        {
            // Timeout
        }
    }
}
```

5.3.12.3 Parameters

AccountOperationSelectionParameters: class
IAccountOperationSelectionResultListener: interface

5.3.12.4 Callback

void **OnSuccess**(AccountOperationSelectionResult result)
void **OnFailure**(AccountOperationSelectionFailureResult result)
void **OnDisplayRequest**(DisplayRequestData requestData)
void **OnInputRequest**(InputRequestData requestData, IInputReceiver inputReceiver)
void **OnInputRequestAborted**(InputRequestAbortedData requestAbortedData)
void **OnPrintRequest**(PrintRequestData requestData, IPrintReceiver printReceiver)
void **OnTimeout**(TimeoutReason reason)

5.4 Parameter objects

5.4.1 VimConfig

5.4.1.1 Description

Object containing parameters regarding VIM configuration

5.4.1.2 VimConfig example

```
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Configuration;

namespace VimNetSamples
{
    class VimConfigDemo
    {
        public VimConfig CreateVimConfig()
        {
            try
            {
                return new VimConfig.Builder()
                    .LogAdditivity(true)
                    .LogLevel(VimLogLevel.DEBUG)
                    .DefaultCurrency(CurrencyType.SEK)
                    .Build();
            }
            catch (VimConfigException e)
            {
                // Parameter validation failed
                return null;
            }
        }
    }
}
```

5.4.1.3 Parameters

LogLocation(string): Optional parameter. Must be writable. On iOS this should be a subfolder of the Documents directory.
LogLevel(VimLogLevel): Optional parameter. VIM log level.
LogAdditivity(bool): Optional parameter. Ignored. See the section about logging for more information.
MaxNumberOfConnectedTerminals(int): Optional parameter. The maximum number of possible connected terminals to the current ECR. Default 1.
TerminalListeningPort(int): Optional parameter. The port on the ECR used for listening for terminal connections. Default 9600.
DefaultEcrLanguage(LanguageType): Optional parameter. Default ECR language. Default English.
DefaultCurrency(CurrencyType): Optional parameter. Default currency used if currency isn't explicitly set for a transaction. Default EUR (Euro).
ConnectionBrokenDetection(boolean): Optional parameter. If set VIM will detect broken connections with terminals based on the absence of keep alive messages. Default true.
DefaultTransactionId(int): Optional parameter. The default transaction id used not not explicitly set for a transaction. Default 1.
DebugMode(bool): Optional parameter. Should only be enabled for test and debugging purposes. If set VIM may force a crash on certain errors. Default false.
ConnectMode(VimConnectMode): Optional parameter. Sets VIM's mode of operation for transport connections used for communication between ECR/VIM and terminal. Default TerminalConnectMode.

5.4.2 LoginParameters

5.4.2.1 Description

Object containing parameters regarding a requested login session on each terminal API instance.

5.4.2.2 LoginParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
   class LoginParametersDemo
   {
      public LoginParameters CreateLoginParameters()
      {
         try
         {
            return new LoginParameters.Builder()
               .EcrId("EcrA")
               .SoftwareManufacturer("Verifone")
               .SoftwareName("MyEcrSoftware")
               .SoftwareVersion("1.2.4 build 8")
               .EcrSerial("MyGlobalUniqueEcrID: 123-456-789")
               .AddEcrCapability(EcrCapabilitiesType.CashierInput)
               .AddEcrCapability(EcrCapabilitiesType.CashierDisplay)
               .AddEcrCapability(EcrCapabilitiesType.CashierError)
               .AddEcrCapability(EcrCapabilitiesType.CashierStatus)
               .AddEcrCapability(EcrCapabilitiesType.PrinterReceipt)
               .AddEcrCapability(EcrCapabilitiesType.SignatureCapture)
               .AddEcrCapability(EcrCapabilitiesType.SignatureVerification)
               .EcrLanguage(LanguageType.English)
               .Build();
         }
         catch (ArgumentException e)
         {
            // Invalid argument provided
            return null;
         }
      }
   }
}
```

5.4.2.3 Parameters

SoftwareManufacturer(string):Mandatory parameter. Name of the manufactorer of the ECR application.
SoftwareName(string): Mandatory parameter. ECR application name.
SoftwareVersion(string): Mandatory parameter. Describes the version of the current ECR application.
EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
EcrSerial(string):Mandatory parameter. Globally unique ID identifying the individual ECR for the ECR software manufacturer. Ex.: "HW ID" or similar.
EcrLanguage(LanguageType): Mandatory parameter. Operator language.
SoftwareCertification(string): Optional parameter. Certification code of the ECR software.
AddEcrCapability(EcrCapabilitiesType): Optional parameter. Specifies the capabilities of the ECR software.

5.4.3 LogoutParameters

5.4.3.1 Description

Object containing parameters regarding a logout on each terminal API instance.

5.4.3.2 LogoutParameters example

```
using System;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class LogoutParametersDemo
    {
        public LogoutParameters CreateLogoutParameters()
        {
            try
            {
                return new LogoutParameters.Builder()
                    .EcrId("EcrA")
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.3.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"

5.4.4 TransactionParameters

5.4.4.1 Purchase

5.4.4.1.1 Description

Object containing the parameters needed for calling StartTransaction() for a regular purchase transaction.

5.4.4.1.2 TransactionParameters purchase example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Token;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class PurchaseParametersDemo
    {
        public TransactionParameters CreatePurchaseParameters()
        {
            try
            {
                return new TransactionParameters.PurchaseBuilder()
                    .EcrId("EcrA")
                    .EcrTransactionId(new TransactionId("123", DateTime.Now))
                    .Amount(100)
                    .Currency(CurrencyType.SEK)
                    .AddTokenRequest(new TokenRequest.Builder()
                                    .Type(TokenType.Customer)
                                    .Build())
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.4.1.3 Parameters

EcrId(string): Mandatory parameter. ECR ID.
EcrTransactionId(TransactionId): Mandatory parameter. ECR transaction ID.
Amount(decimal): Mandatory parameter. Transaction amount.
Currency(CurrencyType): Optional parameter. Transaction currency.
EcrToGatewayData(string): Optional parameter. Additional data sent to gateway. Typically avaliable in transaction reports.
AddTokenRequest(TokenRequest): Optional parameter. Adds a request to receive a token in the transaction result.
ForceOffline(bool): Optional parameter. Flag forcing offline treatment of the transaction. Will only have an affect if the terminal is configured to take this flag into account.

5.4.4.2 Refund

5.4.4.2.1 Description

Object containing the parameters needed for calling StartTransaction() for a refund transaction.

5.4.4.2.2 TransactionParameters refund example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class RefundParametersDemo
    {
        public TransactionParameters CreateRefundParameters()
        {
            try
            {
                return new TransactionParameters.RefundBuilder()
                    .EcrId("EcrA")
                    .EcrTransactionId(new TransactionId("124", DateTime.Now))
                    .Amount(271)
                    .Currency(CurrencyType.SEK)
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.4.2.3 Parameters

EcrId(string): Mandatory parameter. ECR ID.
EcrTransactionId(TransactionId): Mandatory parameter. ECR transaction ID.
Amount(decimal): Mandatory parameter. Total transaction amount.
Currency(CurrencyType): Optional parameter. Transaction currency.
TerminalTransactionId(TransactionId): Optional parameter. ID identifying an original transaction for the terminal.
AddTokenRequest(TokenRequest): Optional parameter. Adds a request to receive a token in the transaction result.

5.4.4.3 Reversal

5.4.4.3.1 Description

Object containing the parameters needed for calling StartTransaction() for a reversal transaction.

5.4.4.3.2 TransactionParameters reversal example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
   class ReversalParametersDemo
   {
      public TransactionParameters CreateReversalParameters(TransactionId lastEcrTransactionId,
                                                            string orgTerminalId,
                                                            TransactionId lastTerminalTransactionId,
                                                            decimal lastTransactionAmount)
      {
         try
         {
            return new TransactionParameters.ReversalBuilder()
               .EcrId("EcrA")
               .EcrTransactionId(lastEcrTransactionId)
               .TerminalId(orgTerminalId)
               .TerminalTransactionId(lastTerminalTransactionId)
               .Amount(lastTransactionAmount)
               .Build();
         }
         catch (ArgumentException e)
         {
            // Invalid argument provided
            return null;
         }
      }
   }
}
```

5.4.4.3.3 Parameters

EcrId(string): Mandatory parameter. ECR ID.
EcrTransactionId(TransactionId): Optional parameter. The ECRs ID for the transaction to reverse.
TerminalId(string): Mandatory parameter. Terminal ID.
TerminalTransactionId(TransactionId): Mandatory parameter. ID identifying the transaction to reverse for the terminal.
Amount(decimal): Mandatory parameter. Amount of the transaction to reverse. Must match the amount of the original transaction.

5.4.4.4 OneTimeReservation

5.4.4.4.1 Description

Object containing the parameters needed for calling StartTransaction() for a non-incremental reservation transaction.

5.4.4.4.2 TransactionParameters OneTimeReservation example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace Verifone.Vim.Tests.code_samples
{
    class OneTimeReservationParametersDemo
    {
        public TransactionParameters CreateOneTimeReservationParameters()
        {
            try
            {
                return new TransactionParameters.OneTimeReservationBuilder()
                    .EcrId("EcrA")
                    .EcrTransactionId(new TransactionId("126", DateTime.Now))
                    .Amount(500)
                    .Currency(CurrencyType.SEK)
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.4.4.3 Parameters

EcrId(string): Mandatory parameter. ECR ID.
EcrTransactionId(TransactionId): Mandatory parameter. ECR transaction ID.
Amount(decimal): Mandatory parameter. Total transaction amount.
Currency(CurrencyType): Optional parameter. Transaction currency.

5.4.4.5 ReservationCompletion

5.4.4.5.1 Description

Object containing the parameters needed for calling StartTransaction() for completing/capturing a reservation transaction.

5.4.4.5.2 TransactionParameters ReservationCompletion example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace Verifone.Vim.Tests.code_samples
{
    class ReservationCompletionParametersDemo
    {
        public TransactionParameters CreateReservationCompletionParameters(
                                                    string reservationTerminalId,
                                                    TransactionId reservationTransactionId)
        {
            try
            {
                return new TransactionParameters.ReservationCompletionBuilder()
                    .EcrId("EcrA")
                    .EcrTransactionId(new TransactionId("127", DateTime.Now))
                    .TerminalId(reservationTerminalId)
                    .TerminalTransactionId(reservationTransactionId)
                    .Amount(453)
                    .Currency(CurrencyType.SEK)
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.4.5.3 Parameters

EcrId(string): Mandatory parameter. ECR ID.
EcrTransactionId(TransactionId): Mandatory parameter. ECR transaction ID.
TerminalId(string): Mandatory parameter if using a different terminal than for the reservation: Terminal ID for the terminal used to perform the reservation.
TerminalTransactionId(TransactionId): Mandatory parameter. The terminals transaction ID for the reservation to be completed.
Amount(decimal): Mandatory parameter. Total transaction amount.
Currency(CurrencyType): Optional parameter. Transaction currency.

5.4.5 AdminParameters

5.4.5.1 Description

Object containing parameters specifying an administrative service.

5.4.5.2 AdminParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class AdminParametersDemo
    {
        public AdminParameters CreateAdminParameters()
        {
            try
            {
                return new AdminParameters.Builder()
                    .EcrId("EcrA")
                    .ServiceIdentification(ServiceIdentificationType.PrintLastReceipt)
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.5.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
ServiceIdentification(ServiceIdentificationType):Mandatory parameter. Identifies the administrative service

5.4.6 ReconciliationParameters

5.4.6.1 Description

Object containing parameters specifying a reconciliation type.

5.4.6.2 ReconcilicationParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class ReconciliationParametersDemo
    {
        public ReconciliationParameters CreateReconciliationParameters()
        {
            try
            {
                return new ReconciliationParameters.Builder()
                    .EcrId("EcrA")
                    .ReconciliationType(ReconciliationType.AcquirerReconciliation)
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.6.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
ReconciliationType(ReconciliationType):Mandatory parameter. Identifies the reconciliation type

5.4.7 TransactionStatusParameters

5.4.7.1 Description

Object containing parameters needed for calling transactionStatus().

5.4.7.2 TransactionStatusParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class TransactionStatusParametersDemo
    {
        public TransactionStatusParameters CreateTransactionStatusParameters(
            TransactionStatusType orgTransactionType,
            string orgTransactionServiceId)
        {
            try
            {
                return new TransactionStatusParameters.Builder()
                        .EcrId("EcrA")
                        .ReferenceServiceId(orgTransactionServiceId)
                        .TransactionStatusType(orgTransactionType)
                        .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.7.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
TransactionStatusType(TransactionStatusType):Mandatory parameter. Type of the original transaction.
ReferenceServiceId(string):Mandatory parameter. Service id of the original transaction

5.4.8 AbortParameters

5.4.8.1 Description

Object containing the parameters needed to call abort().

5.4.8.2 AbortParameters example

```
using System;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class AbortParametersDemo
    {
        public AbortParameters CreateAbortParameters()
        {
            try
            {
                return new AbortParameters.Builder()
                    .EcrId("EcrA")
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.8.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"

5.4.9 InputParameters

5.4.9.1 Description

Object containing parameters specifying a request for input from the terminal.

5.4.9.2 InputParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class InputParametersDemo
    {
        public InputParameters CreateInputParameters()
        {
            try
            {
                return new InputParameters.Builder()
                        .EcrId("EcrA")
                        .InputType(InputType.Barcode)
                        .TimeoutInSeconds(5)
                        .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.9.3 Parameters

EcrId(string): Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
InputType(InputType): Mandatory parameter. Type of input requested.
TimeoutSeconds(int): Optional parameter. The number of seconds the terminal will wait for input before timing out and sending a negative response.

5.4.10 CardInfoParameters

5.4.10.1 Description

Object containing parameters for getCardInfo.

5.4.10.2 CardInfoParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Loyalty;
using Verifone.Vim.Api.Common.Token;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class CardInfoParametersDemo
    {
        public CardInfoParameters CreateCardInfoParameters()
        {
            try
            {
                return new CardInfoParameters.Builder()
                        .EcrId("EcrA")
                        .EcrTransactionId(new TransactionId("126", DateTime.Now))
                        .LoyaltyHandling(LoyaltyHandlingType.Allowed)
                        .AddTokenRequest(new TokenRequest.Builder()
                                    .Type(TokenType.Customer)
                                    .SchemeId("MyTokenSchemeID")
                                    .Build())
                        .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.10.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
EcrTransactionId(TransactionId): Mandatory parameter. ECR transaction id.
LoyaltyHandling(LoyaltyHandlingType): Optional parameter. Type of loyalty processing requested by the ECR
AddTokenRequest(TokenRequest): Optional parameter. Adds a request to receive a token in the transaction result.

5.4.11 ResetContextParameters

5.4.11.1 Description

Object containing parameters for resetContext.

5.4.11.2 ResetContextParameters example

```
using System;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class ResetContextParametersDemo
    {
        public ResetContextParameters CreateResetContextParameters()
        {
            try
            {
                return new ResetContextParameters.Builder()
                    .EcrId("EcrA")
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.11.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"

5.4.12 PrintParameters

5.4.12.1 Description

Object containing parameters specifying a request to print on a payment terminal.

5.4.12.2 PrintParameters example

```
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Print;
using Verifone.Vim.Api.Common.TextFormatting;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class PrintParametersDemo
    {
        public PrintParameters CreatePrintParameters()
        {
            return new PrintParameters.Builder()
                    .EcrId("EcrA")
                    .AddPrintElement("Simple text line")
                    .AddPrintElement(new TextPrintElement.Builder()
                        .Text("Formatted text line")
                        .TextStyle(TextStyle.Bold)
                        .TextAlignment(TextAlignment.Right)
                        .TextHeight(TextHeight.DoubleHeight)
                        .TextWidth(TextWidth.DoubleWidth)
                        .Build())
                    .AddPrintElement(new TextPrintElement.Builder()
                        .Text("Partial line ")
                        .EndOfLineFlag(false)
                        .Build())
                    .AddPrintElement(new TextPrintElement.Builder()
                        .Text("Rest line")
                        .TextStyle(TextStyle.Bold)
                        .Build())
                    .AddPrintElement(new LinearBarcodePrintElement.Builder()
                        .Type(BarcodeType.EAN8)
                        .Value("02200220")
                        .Build())
                    .AddPrintElement("")
                    .AddPrintElement(new LinearBarcodePrintElement.Builder()
                        .Type(BarcodeType.Code128)
                        .Value("test12345")
                        .Height(100)
                        .Width(300)
                        .Build())
                    .AddPrintElement("")
                    .AddPrintElement(new QrCodePrintElement.Builder()
                        .Value("verifone.com")
                        .Height(300)
                        .Width(300)
                        .ErrorCorrection(QrCodeErrorCorrectionType.High)
                        .Build())
                    .AddPrintElement("")
                    .AddPrintElement(new AztecCodePrintElement.Builder()
                        .Value("verifone.com")
                        .Height(300)
                        .Width(300)
                        .Build())
                    .AddPrintElement("")
                    .AddPrintElement("")
                    .AddPrintElement("")
                    .Build();
        }
    }
}
```

5.4.12.3 Parameters

EcrId(string):Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"
AddPrintElement(string): Optional. Convenience method adding a simple text to the list of elements to be printed on the terminal. At least one element must be added for the parameters to be valid.
AddPrintElement(PrintElement): Optional. Adding a PrintElement representing a formatted text or barcode to the list of elements to be printed on the terminal. At least one element must be added for the parameters to be valid.

5.4.13 BalanceInquiryParameters

5.4.13.1 Description

Object containing parameters for balance inquiry.

5.4.13.2 BalanceInquiryParameters example

```
using System;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
   class BalanceInquiryParametersDemo
   {
      public BalanceInquiryParameters CreateBalanceInquiryParameters()
      {
         try
         {
            return new BalanceInquiryParameters.Builder()
                  .EcrId("EcrA")
                  .Build();
         }
         catch (ArgumentException e)
         {
            // Invalid argument provided
            return null;
         }
      }
   }
}
```

5.4.13.3 Parameters

EcrId(string): Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"

5.4.14 AccountSelectionParameters

5.4.14.1 Description

Object containing parameters for StartAccountSelecion.

5.4.14.2 AccountSelectionParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Account;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class AccountSelectionParametersDemo
    {
        public AccountSelectionParameters CreateAccountSelectionParameters()
        {
            try
            {
                return new AccountSelectionParameters.Builder()
                    .EcrId("EcrA")
                    .AddAccountItem(new AccountItem.Builder()
                            .AccountId(new AccountId("12#1", accountTimestamp1))
                            .Status(AccountStatusType.Open)
                            .OperatorId("Operator1")
                            .Currency(CurrencyType.SEK)
                            .TotalAmount(2088.66m)
                            .RemainingAmount(1001.01m)
                            .Build())
                    .AddAccountItem(new AccountItem.Builder()
                            .AccountId(new AccountId("12#2", accountTimestamp2))
                            .Status(AccountStatusType.Open)
                            .OperatorId("Operator1")
                            .Currency(CurrencyType.SEK)
                            .TotalAmount(557.12m)
                            .Build())
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.14.3 Parameters

EcrID(string): Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"

AddAccountItem(AccountItem): Optional parameter. Identifies an account, contains basic information needed to facilitate account selection.

5.4.15 AccountOperationSelectionParameters

5.4.15.1 Description

Object containing parameters for StartAccountOperationSelecion.

5.4.15.2 AccountOperationSelectionParameters example

```
using System;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Common.Account;
using Verifone.Vim.Api.Common.PaymentInstrumentData;
using Verifone.Vim.Api.Parameters;
using Verifone.Vim.Api.Results;

namespace VimNetSamples
{
    class AccountOperationSelectionParametersDemo
    {
        public AccountOperationSelectionParameters CreateAccountOperationSelectionParameters()
        {
            try
            {
                return new AccountOperationSelectionParameters.Builder()
                    .EcrId("EcrA")
                    .AccountItem(new AccountItem.Builder()
                            .AccountId(new AccountId("12#1", accountTimestamp1))
                            .Status(AccountStatusType.Closed)
                            .OperatorId("Operator1")
                            .Currency(CurrencyType.SEK)
                            .TotalAmount(2088.66m)
                            .RemainingAmount(0)
                            .Build())
                    // Add failed account operation results
                    // Also see onFailure in StartTransaction example
                    .AddAccountOperationResult(new AccountOperationResult.Builder()
                            .AccountOperation(AccountOperationType.PartialPayment)
                            .TransactionFailureResult(new TransactionFailureResult.Builder()
                                    .EcrId("EcrA")
                                    .TerminalId("123-456-789")
                                    .ServiceId("123")
                                    .Error(FailureErrorType.Cancel)
                                    .AdditionalReason("User cancellation during purchase processing")
                                    .EcrTransactionId(new TransactionId("580", ecrTransactionTimestamp1))
                                    .TerminalTransactionId(new TransactionId("322", transactionTimestamp1))
                                    .PaymentInstrumentData(new PaymentInstrumentData.Builder()
                                            .PaymentInstrumentType(PaymentInstrumentType.Card)
                                            .CardData(new CardData.Builder()
                                                    .PaymentBrand("MASTERCARD")
                                                    .MaskedPan("444444******1111")
                                                    .EntryMode(EntryModeType.ICC)
                                                    .Build())
                                            .Build())
                                    .Build())
                            .Build())
                    // Add successful account operation results
                    // Also see onSuccess in StartTransaction example
                    .AddAccountOperationResult(new AccountOperationResult.Builder()
                            .AccountOperation(AccountOperationType.CompletePayment)
                            .TransactionResult(new TransactionResult.Builder()
                                    .EcrId("EcrA")
                                    .TerminalId("123-456-789")
                                    .ServiceId("124")
                                    .EcrTransactionId(new TransactionId("581", ecrTransactionTimestamp2))
                                    .TerminalTransactionId(new TransactionId("323", transactionTimestamp2))
                                    .PaymentInstrumentData(new PaymentInstrumentData.Builder()
                                            .PaymentInstrumentType(PaymentInstrumentType.Cash)
                                            .Build())
                                    .AuthorizedAmount(2088.66m)
                                    .Build())
                            .Build())
                    .CurrentAccountOperationResult(new CurrentAccountOperationResult.Builder()
                            .AccountOperation(AccountOperationType.CompletePayment)
                            .TerminalTransactionId(new TransactionId("323", transactionTimestamp2))
                            .Build())
                    .Build();
            }
            catch (ArgumentException e)
            {
                // Invalid argument provided
                return null;
            }
        }
    }
}
```

5.4.15.3 Parameters

EcrID(string): Mandatory parameter. Logical ID, describing a locally unique identification of the ECR typically used for support references. Ex.: "Register1"

AccountItem(AccountItem): Mandatory parameter. Identifies an account.

AddAccountOperationResult(AccountOperationResult): Optional parameter. Detailed information about the result of an account operation in a pay at table system.

CurrentAccountOperationResult(CurrentAccountOperationResult): Optional parameter. Identifies an account operation that was just finished and requires post handling. References the the detailed information about the previous financial operations for an account.

5.5 Input requests and authentication methods

Methods handling input requests from the terminal. This includes requests for email address, password or electronic signature, which allows the ECR to provide information when prompted by the terminal.

Input type	Format	Description
Text	string	Text input.
Password	string	Password input. For example, used by terminal to request password before starting reversal or refund transactions.
Email	string	Email address input. For example, used by terminal to request email for sending e-receipt.
SignatureCapture	PNG image as byte[]	Electronic signature capture (handwritten signature).
Confirmation	ConfirmType	Yes/No input

5.5.1 OnInputRequest example

```
using System.Drawing;
using System.IO;
using Verifone.Vim.Api.Common.SignatureCapture;
using Verifone.Vim.Api.DeviceRequests.Input;

namespace VimNetSamples
{
    class OnInputRequestDemo
    {
        Image signatureImage;

        public void OnInputRequest(InputRequestData requestData, IInputReceiver inputReceiver)
        {
            InputRequestType inputType = requestData.InputType;

            switch (inputType)
            {
                 case InputRequestType.Text:
                    inputReceiver.InputText("some text 123");
                    break;
                case InputRequestType.SignatureCapture:
                    byte[] imageInByte = ImageToByteArray(signatureImage);
                    inputReceiver.InputSignature(new Signature.Builder()
                        .ImageFormat(ImageFormatType.PNG)
                        .ImageData(imageInByte)
                        .Verified(true)
                        .Build());
                    break;
                case InputRequestType.Email:
                    inputReceiver.InputText("myEmailAccount@gmail.com");
                    break;
                case InputRequestType.Password:
                    inputReceiver.InputText("myPassword123");
                    break;
                case InputRequestType.Confirmation:
                    inputReceiver.InputConfirmation(ConfirmType.Yes);
                    break;
                default:
                    // Unhandled user input request
                    inputReceiver.InputCancel();
                    break;
            }
        }

        public byte[] ImageToByteArray(Image imageIn)
        {
            byte[] result = null;
            using (MemoryStream ms = new MemoryStream())
            {
                imageIn.Save(ms, System.Drawing.Imaging.ImageFormat.Png);
                result = ms.ToArray();
            }
            return result;
        }
    }
}
```

5.5.2 Notes

Note that the customer signature image needs to be in PNG format and cannot exceed 22500 bytes. The signature should be manually verified by the operator and the verification result passed as a boolean in inputSignature().

5.6 Transaction identification

Every transaction is identified by the ECR and payment terminal. In addition an acquirer transaction ID is available. This can be used to trace a transaction through the whole payment system. Each transaction ID are data structures with the type TransactionId, which includes two data elements that together provides a unique identification:

1. A string id, to identify the transaction for the terminal or the ECR.
2. The date and time of the transaction as a Date timestamp, allowing the uniqueness of the identification.

VIM operates with three separate transaction IDs:

- **ECR transaction ID** - Assigned to the transaction by the ECR software and returned in the transaction reponse from the payment terminal.
- **Terminal transaction ID** - Assigned to the transaction by the payment terminal and returned in the transaction response. The string ID part is a globally unique identificator (UUID). May be used to identify the transaction info available to the ECR software with transaction reports available from the Verifone gateway.
- **Acquirer transaction ID** - The string ID part of the Acquirer transaction ID contains the Retrieval Reference Number (RRN). Together with the Terminal ID, it may be used to trace a transaction through the whole payment system.

5.6.1 Transaction identification code sample

The ECR may assign its own transaction ID or it may be generated in VIM before the start of the transaction and passed along in TransactionParameters like this:

```
TransactionId currEcrTransactionID = new TransactionId("myTransactionId-123", new DateTime());
TransactionParameters parameters = new TransactionParameters.PurchaseBuilder()
            .EcrTransactionId(currEcrTransactionID)
            ...
            .Build();
```

The unique terminal TransactionID will be generated during transaction handling on the terminal.

Both will be provided by the terminal as a parameter in the transaction result and may be fetched like this:

```
public void OnSuccess(TransactionResult result)
{
   TransactionId ecrTransId = result.EcrTransactionId;
   TransactionId terminalTransId = result.TerminalTransactionId;
}
```

5.7 Sensitive card data

By accessing *TransactionResult → PaymentInstrumentData → CardData → SensitiveCardData* one has seemingly access to all the sensitive data from the card (full pan, track data etc).The payment terminal will never send actual sensitive card data to the ECR. The API is there to allow the payment terminal to send all the card data to the ECR for white listed non-PCI cards. This functionality is intended to allow a cashier to login to the ECR swiping a non-sensitive “cashier card” on the payment terminal, reading local bonus card schemes etc.

5.8 Use cases

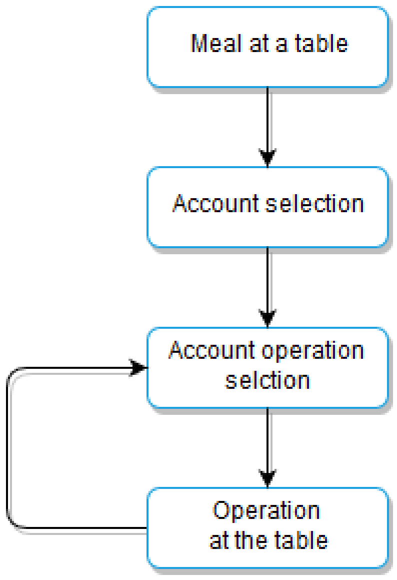
5.8.1 Pay at table

"Pay at table" is a typical use case for payment at a restaurant where:

- An account is created and maintained per table for the period a party of guests stays at the restaurant.
- The account is often settled by multiple payments by several people at the table.
- Multiple payment means, including cash, are often used.
- The payment is often split between the guests depending on their consumption.
- Tips is frequently offered.
- Payment happens at the table using a portable terminal operated by a waiter.
- The portable terminal communicates with a central system to coordinate the settlement of the account for a table (i.e. a regular ECR system for starting transactions, receipt printing, etc. is not readily available).

The process of the "pay at table" includes following steps:

- Meal at a table: account for a “table” is created.
- Account selection: The waiter takes a portable terminal and types an account search string to allow the terminal to retrieve matching accounts from central system. The waiter selects desired account.
- Account operation selection: The waiter selects an account operation to perfom.
- Operation at the table: Payments or non-financial operations are processed at the table. The waiter is prompted to select new operations until the account is successfully settled (remaining amount is zero) or processing is canceled.



Account operations are initiated by the waiter. VIM will notify central system that account selection has been started with *OnAccountSearchEvent(AccountSearchEvent evnt)* callback (see chapter 5.1). *AccountSearchCriteria* is used to transfer additional information for account matching by central system. When matching is performed central system can use *StartAccountSelection* (see chapter 5.3.11) to transfer information about a list of available accounts to the payment terminal and facilitate the selection of an account for further processing. If only one account matches search criteria provided by terminal, the central system can skip account selection and use *StartAccountOperationSelection* (see chapter 5.3.12) to allow the terminal to request an account operation to be started. The following operations are currently available:

- Complete payment
- Partial payment
- Print receipt
- Print account details

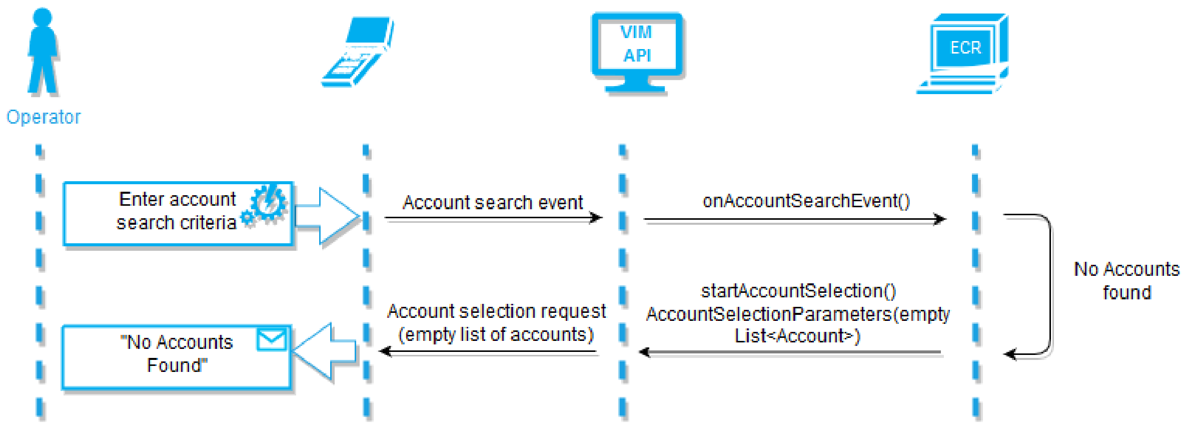
The central pay at table system must ensure the creation and maintenance of accounts (status, remaining amount, etc.) and storage of account operation results. To ensure that duplicate or overlapping payments are not made, only one terminal should be able to process financial operations for an account at a time. The central pay at table system is responsible for performing account locking to enforce this. Non-financial operations such as printing should be possible for all accounts without regard to their status.

5.8.1.1 Pay at table flow charts

The flow charts in this section describe some of the standard pay at table flows. The purpose is to visualize the processes and to make them easier to implement for integrators. Details of terminal transaction processing are left out to provide a clear picture.

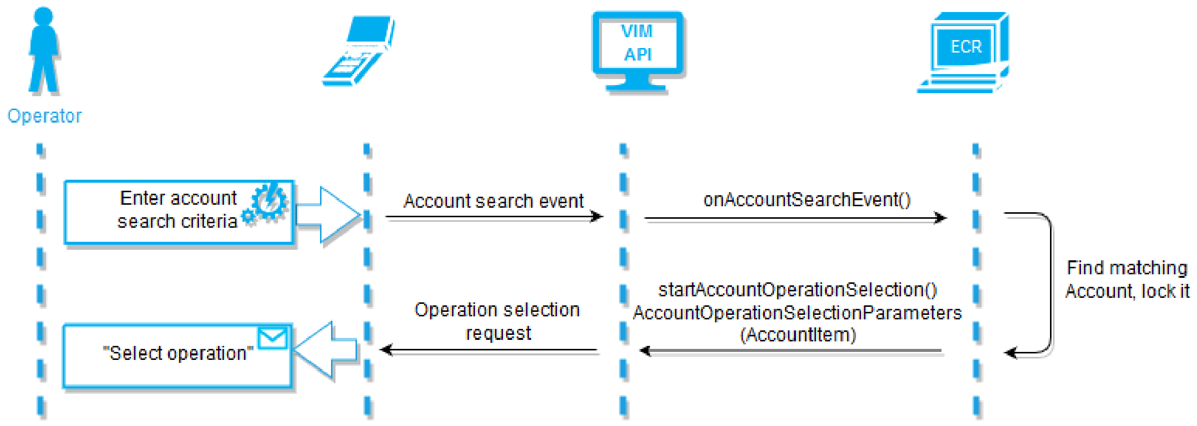
5.8.1.1.1 Account selection - no accounts found

This flow chart describes an account selection when the search has ended with no results. After error message is shown, the operator can start a new search request.


Operator
VIM API
ECR
Enter account search criteria
Account search event
onAccountSearchEvent()
No Accounts found
startAccountSelection() AccountSelectionParameters(empty List<Account>)
Account selection request (empty list of accounts)
"No Accounts Found"

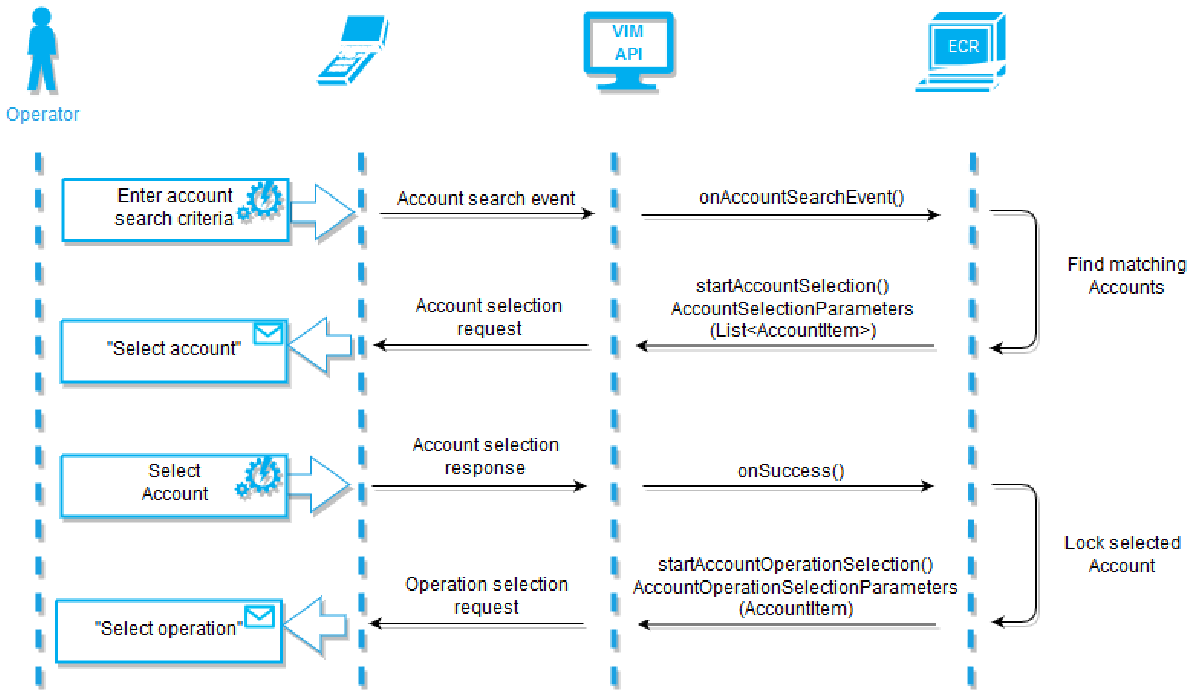
5.8.1.1.2 Account selection - single account found

This flow chart describes an account selection when the search has ended with single result. Manual account selection is skipped, operator is immediately prompted to select an account operation. Flow continues based on selected operation (see chapter 5.8.1.1.4, 5.8.1.1.5, 5.8.1.1.6)


Operator
VIM API
ECR
Enter account search criteria
Account search event
onAccountSearchEvent()
Find matching Account, lock it
startAccountOperationSelection()
AccountOperationSelectionParameters
(AccountItem)
Operation selection request
"Select operation"

5.8.1.1.3 Account selection - multiple accounts found

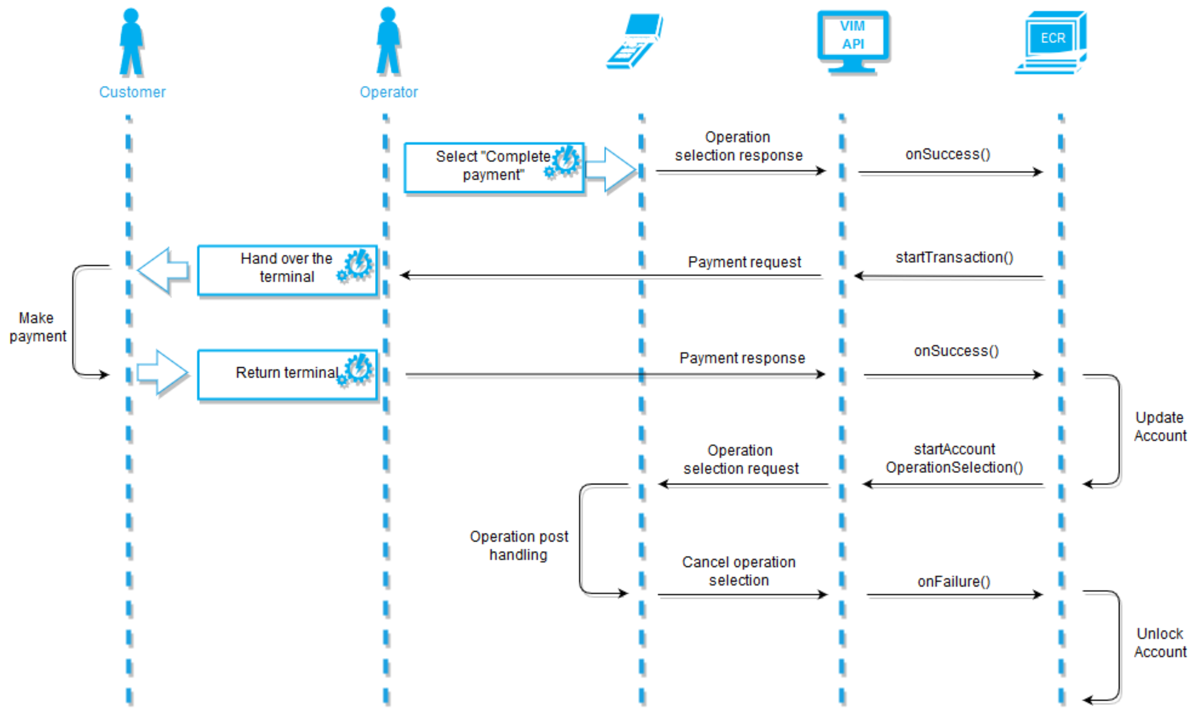
This flow chart describes an account selection when the search has ended with multiple results. After operator has selected an account from the list, he is prompted to select an account operation. Flow continues based on selected operation (see chapter 5.8.1.1.4, 5.8.1.1.5, 5.8.1.1.6)



Operator
VIM API
ECR
Enter account search criteria
Account search event
onAccountSearchEvent()
Find matching Accounts
"Select account"
Account selection request
startAccountSelection() AccountSelectionParameters (List<AccountItem>)
Select Account
Account selection response
onSuccess()
Lock selected Account
"Select operation"
Operation selection request
startAccountOperationSelection() AccountOperationSelectionParameters (AccountItem)

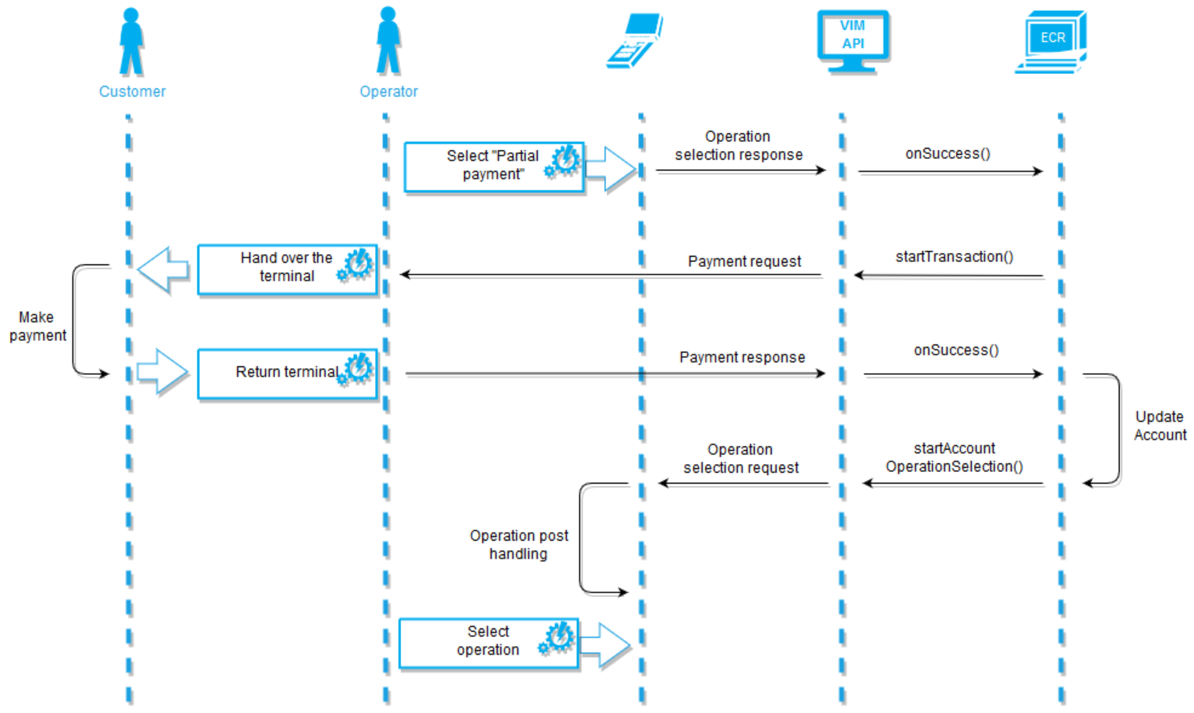
5.8.1.1.4 Account operation selection - complete payment

This flow chart describes a complete payment on selected account. After payment is completed, remaining amount is zero and account is considered settled. The operator can start a new search request (see chapter 5.8.1.1.1, 5.8.1.1.2, 5.8.1.1.3).



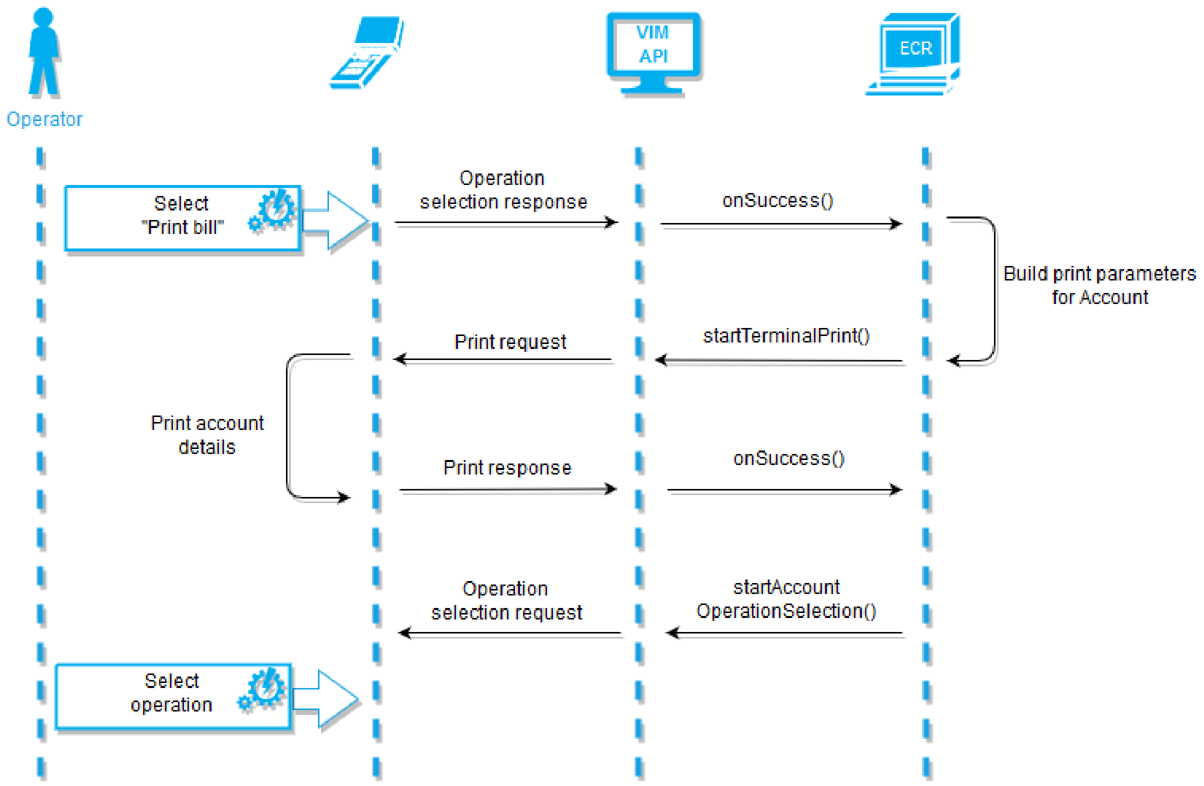
5.8.1.1.5 Account operation selection - partial payment

This flow chart describes a partial payment on selected account. After payment is completed, operator can select next operation to perform, flow continues based on selected operation (see chapter 5.8.1.1.4, 5.8.1.1.5, 5.8.1.1.6).



5.8.1.1.6 Account operation selection - print account details

This flow chart describes print of an account details on selected account. After print is completed, operator can select next operation to perform, flow continues based on selected operation (see chapter 5.8.1.1.4, 5.8.1.1.5, 5.8.1.1.6).



6 Logging

VIM has a strict dependency on logging being present for support purposes, and Verifone may refuse to provide any support if the customer cannot provide VIM logs or they are not implemented properly.

Logging in VIM.net use the NLog logging framework. For the logging functionality to work, the application must configure VIM to store the logs in a suitable location with sufficient access rights. The config parameter LogLocation can be used to specify the full path of the log directory.

Loglevel is recommended set to VimLogLevel.DEBUG to get full context for support cases. (The extra info provided with VimLogLevel TRACE or ALL will only be useful in special cases). It is strongly recommended to test retrieval of VIM logs as soon as possible.

7 Error handling

VIM errors will be passed through specific error events and should be handled in their respective error handlers. Example:

```
public void OnVimError(VimErrorEvent evnt)
{
   // Error handling
}
```

7.1 Receipt copy of the last transaction

For unexpected stops, loss of communication with payment terminal, ECR hangs and timeouts, the ECR should provide functionality to make it easy for the operator to control the result of the last payment transaction. The ECR should implement the possibility to request a receipt copy from the payment terminal. This command will result in a print of the last approved transaction made by the payment terminal. This is an administrative command and is only printing the receipt copy, no result data from the card transaction is returned to the ECR.

There may be more than one transaction with the same amount, and it is important to provide good functionality for manual control of the transaction receipt after using this function, to make sure that the transaction result is from the card of the actual customer and not from the person in front of him in the queue.

The operator should control:

- Date and time of the transaction
- Transaction type
- Amount
- Card type and the last digits of the card number

In order to make a correct record of the card transaction in the ECR the functionality for getting the result of the last transaction should be used.

7.1.1 Receipt copy code sample

7.1.1.1 Description

Starts a request for the last receipt from the payment terminal.

7.1.1.2 StartAdmin with print of last receipt example

```
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.Parameters;

namespace VimNetSamples
{
    class ReceiptCopyDemo
    {
        IVimApi terminalApi;
        IAdminResultListener resultListener;

        public void GetReceiptCopy()
        {
            AdminParameters adminParameters = new AdminParameters.Builder()
                .EcrId("EcrA")
                .ServiceIdentification(ServiceIdentificationType.PrintLastReceipt)
                .Build();
            terminalApi.StartAdmin(adminParameters, resultListener);
        }
    }
}
```

8 Obtaining and verifying the last terminal transaction status

One of the basic rules for an ECR with an integrated payment terminal is to make sure that the result of the card payment transaction is always received and handled correctly by the ECR. However there may be situations where the result of the card payment transaction does not reach the ECR, for instance if the ECR hangs, or if the communication is lost between the ECR and the payment terminal or if the ECR timeout value is too short. If any of these situations occurs, the card transaction may be approved by host and the payment may not be registered in the ECR. If the status of this last transaction is not verified and handled properly, this can lead to the vendor not receiving money for his goods or clients being charged twice.

8.1 Result of last transaction

The ECR may request the result of the last payment transaction from the payment terminal with a transactionStatus call.

The referenceServiceId field required to successfully build a TransactionStatusParameters instance is the ServiceID of the original transaction. This information is available to the ECR software from the parameter object used to start the original transaction (through the method getServiceId() or similar). Basically the ECR software needs to remember the ServiceID of the last transaction it has started in case it needs to check the status of the transaction using transactionStatus.

When making a transactionStatus call the ECR software need to specify referenceServiceId to reference the transaction it wants to check the status for (the original transaction). So the ServiceID used as referenceServiceId should be taken from the parameters object of the original transaction. If the original transaction is a purchase the ServiceID should be collected from the TransactionParameters instance with getServiceId(), and remembered by the ECR software in case a transactionStatus call is needed. This is useful if a communication break occurs during transaction processing, for an instance.

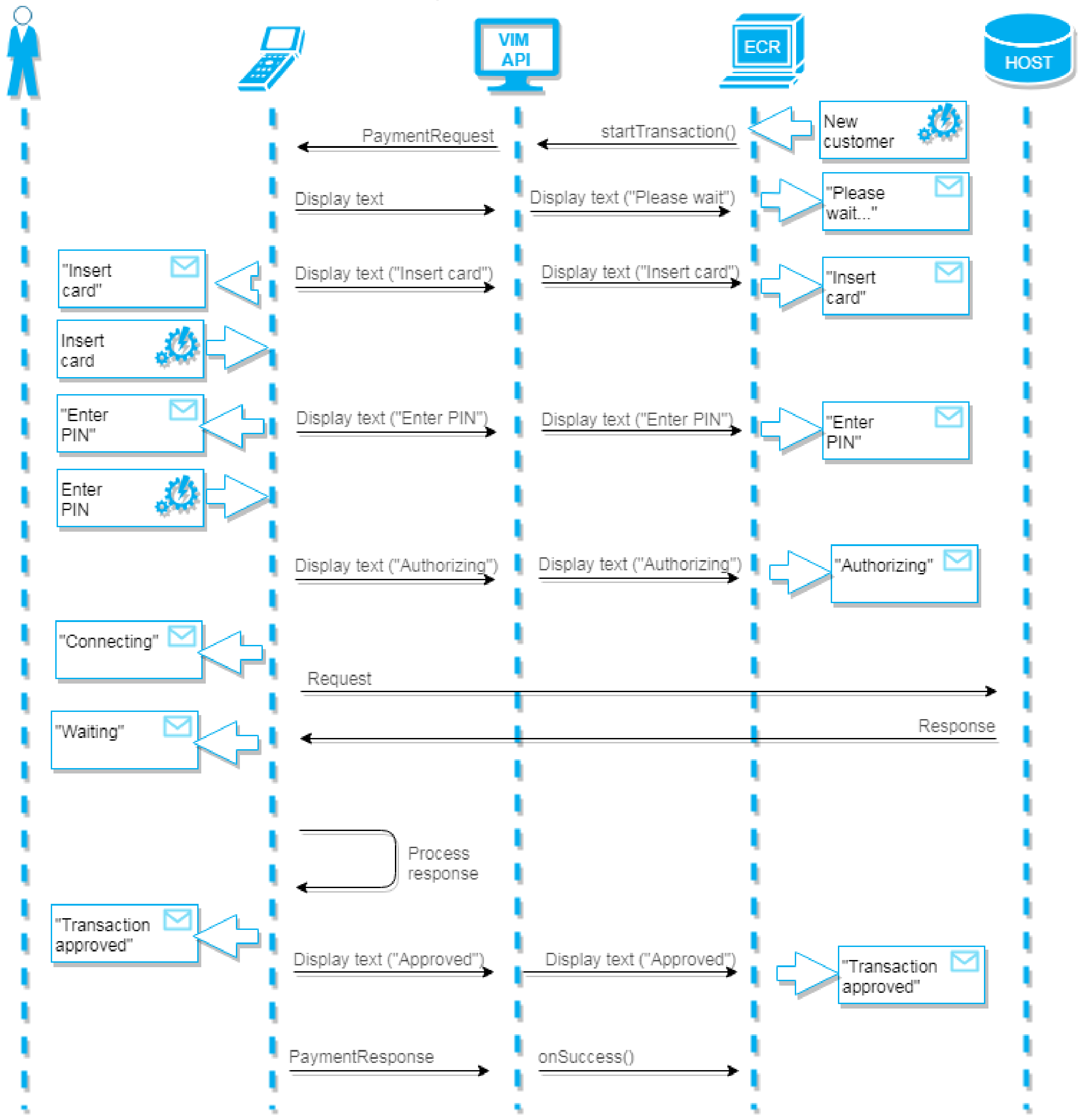
See code sample in chapter 5.3.4.

9 Appendix A:Transaction flow charts

This appendix contains flow charts that describe some of the standard transaction flows. The purpose is to visualize the processes and to make them easier to implement for integrators.

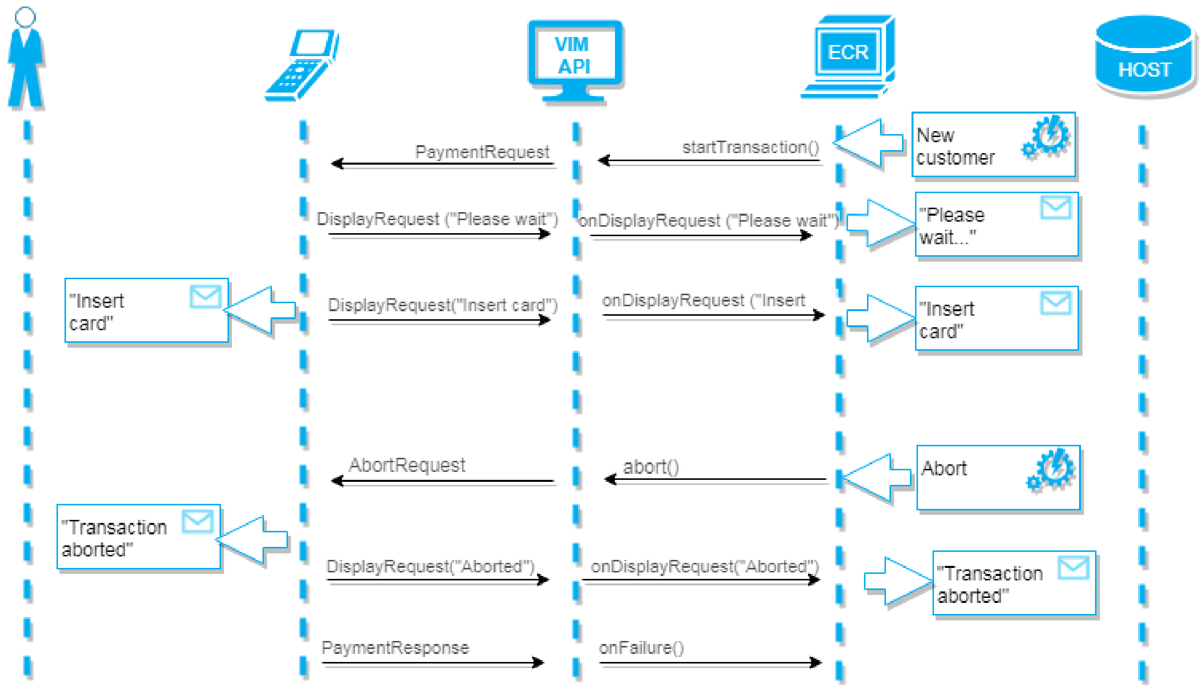
9.1 Purchase

This flow chart describes a standard purchase with PIN.



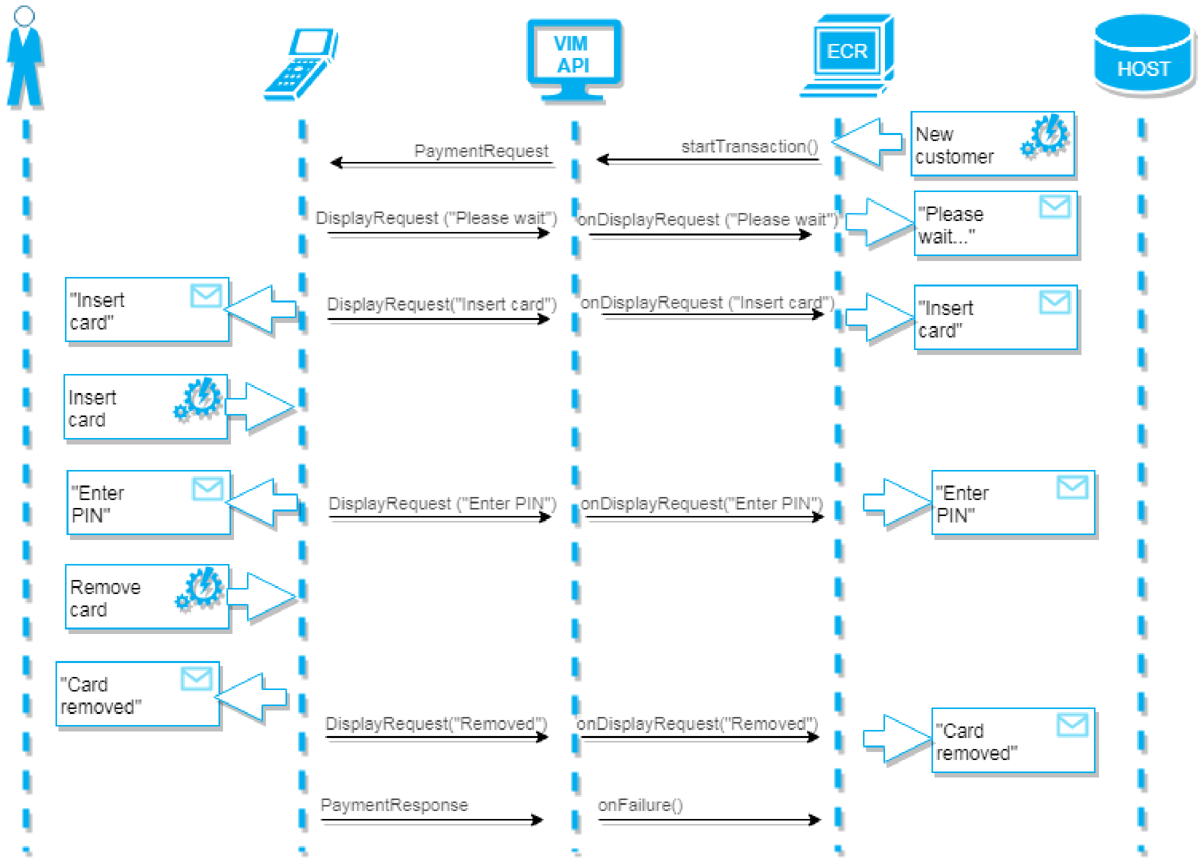
9.2 Aborted purchase (by operator)

This flow chart describes a purchase that has been aborted by operator.



9.3 Aborted purchase (by customer)

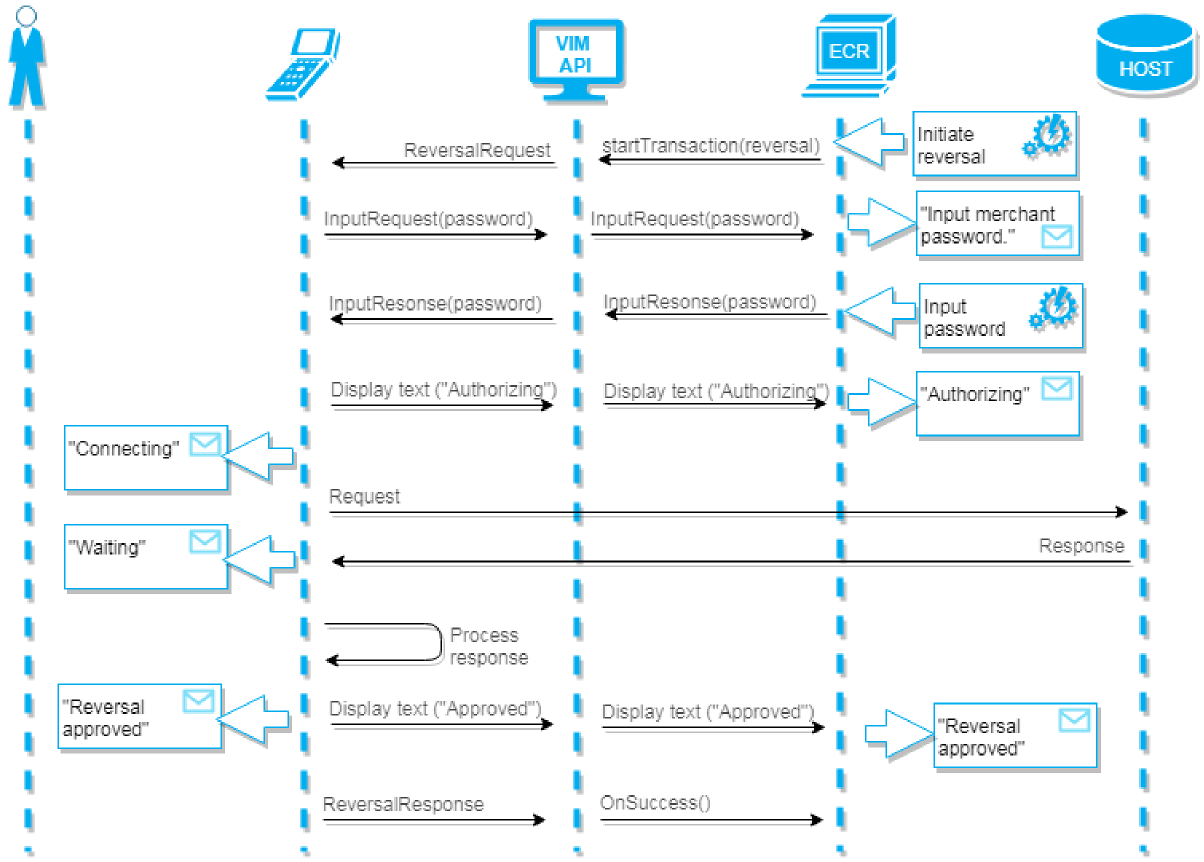
This flow chart describes a purchase that has been aborted by customer.



VIM API
ECR
HOST
New customer
startTransaction()
PaymentRequest
DisplayRequest ("Please wait")
onDisplayRequest ("Please wait")
"Please wait..."
"Insert card"
DisplayRequest("Insert card")
onDisplayRequest ("Insert card")
"Insert card"
Insert card
"Enter PIN"
DisplayRequest ("Enter PIN")
onDisplayRequest("Enter PIN")
"Enter PIN"
Remove card
"Card removed"
DisplayRequest("Removed")
onDisplayRequest("Removed")
"Card removed"
PaymentResponse
onFailure()

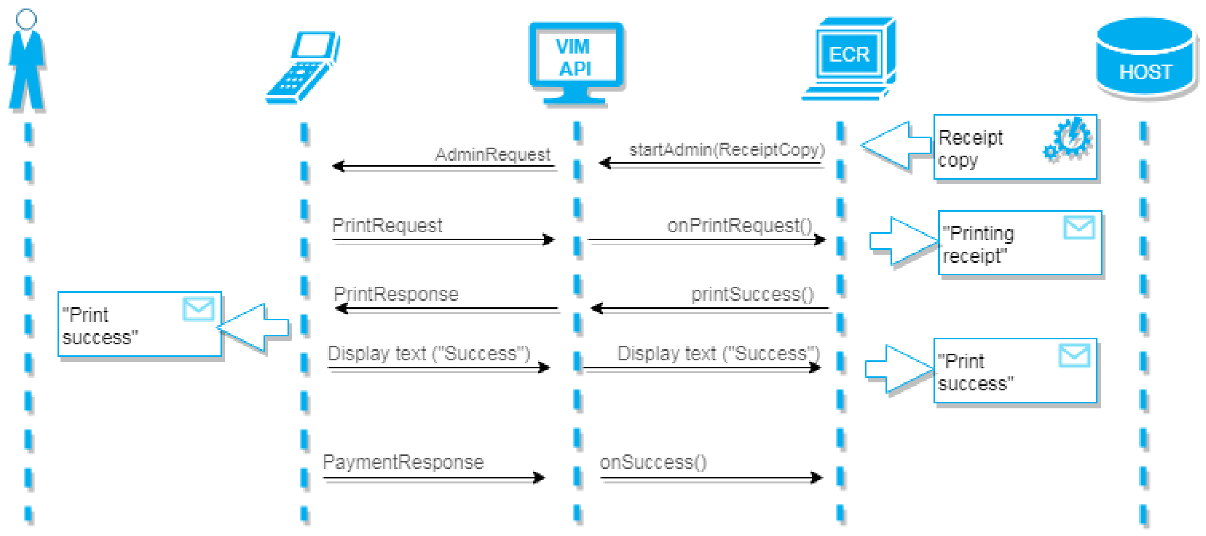
9.4 Reversal

This flow chart describes a reversal of a previous transaction



9.5 Copy of previous receipt

The flow chart describes the communication between VIM and payment terminal to send a copy of the receipt of the latest performed transaction to a specified email address.



10 Appendix B: Connecting to payment terminals

VIM’s normal mode of operation is to only accept incoming connections from payment terminals (*TerminalConnectMode*), but it’s also possible to configure VIM to only allow ECR/VIM to connect to terminals (*EcrConnectMode*) or simultaneous allow both modes of operations for different terminals (*MixedConnectMode*).

VIM’s mode of operation is configured by using the optional *ConnectMode(VimConnectMode)* method when creating the *VimConfig*-instance used during VIM initialisation. See chapter 5.4.1 and 5.1 for more information.

10.1 TerminalConnectMode

When using *TerminalConnectMode,* which is the default, VIM will start listening for terminal connections on a specified port (default is 9600). The payment terminal(s) must be configured to connect to ECR/VIM using the correct IP-address and port.

If the connection goes down for some reason, it’s the payment terminal’s responsibility to reestablish the connection as soon as possible. VIM will notify the ECR software about connection changes with *onConnectionChangedEvent(ConnectionChangedEvent)*-callbacks (see chapter 5.1). Since a payment terminal cannot know when an ECR wants to start an operation, it will strive to always keep the connection open (*ConnectionPersistent*).

10.2 EcrConnectMode

When using *EcrConnectMode* the payment terminal must be configured to accept connections from the ECR. This is not the default operation mode for VEPPNB terminals and this terminal configuration can currently only be made by Verifone.

The ECR software must also provide the information required for VIM to connect to a payment terminal. This is done using the *CreateTerminal(TerminalInformation)*-method on the *Vim* instance after initialisation. The ECR software must first construct a *TerminalInformation* instance providing at least the payment terminal serial number, connection details (IP-address, port) as well as information about how the connection shall be managed.

After successfully creating a terminal the ECR software will receive an *OnTerminalReady(TerminalReadyEvent)* callback as usual, but this doesn’t mean a connection to the payment terminal has been established. The connection will first be established when the ECR software attempts a *Login*.

VIM has two options for managing a terminal connection. The connection with the payment terminal can be always kept open (*ConnectionPersistent*) or taken up and down for each session (*ConnectionPerSession*). A session is defined as the interval between a *Login* and a *Logout* (see chapter 5.2 for more information about session management).

If the connection goes down for some reason or a connection cannot be established in the first place, VIM will notify the ECR softwareabout the changes with *onConnectionChangedEvent(ConnectionChangedEvent)*-callbacks (see chapter 5.1), but in *EcrConnectMode* it's the ECR software's responsibility to reestablish a connection. The ECR software can attempt to reestablish a connection with the payment terminal by performing a *Login* (see chapter 5.2.1).

10.2.1 EcrConnectMode example code

```
using System;
using System.IO;
using System.Net;
using Verifone.Vim.Api;
using Verifone.Vim.Api.Common;
using Verifone.Vim.Api.Configuration;
using Verifone.Vim.Api.Events;
using Verifone.Vim.Api.Listeners;
using Verifone.Vim.Api.TerminalInformation;
using Verifone.Vim.Api.TerminalInformation.TerminalConnection;

namespace VimNetSamples
{
   class EcrConnectModeInitialisationDemo : ITerminalConnectListener, IVimListener,
IVimApiListener
   {
      IVim vim;
      IVimApi terminalApi;

      public void InitialiseVim()
      {
         // Create VIM instance
         try
         {
            vim = VimFactory.CreateVim(new VimConfig.Builder()
               .ConnectMode(VimConnectMode.EcrConnectMode)
               .LogLocation(Directory.GetCurrentDirectory())
               .LogLevel(VimLogLevel.DEBUG)
               .DefaultCurrency(CurrencyType.SEK)
               .Build());

         }
         catch (VimConfigException e)
         {
            // Unable to configure VIM
         }

         // Add mandatory VimListener
         vim.AddVimListener(this);

         // Initialise VIM
         try
         {
            vim.Initialise();
         }
         catch (InvalidOperationException e)
         {
            // Initialisation failed
         }
```

```
            try
            {
                // Construct TerminalInformation instance
                TerminalInformation terminalInformation = new TerminalInformation.Builder()
                    .SerialNumber("123-456-789")
                    .TerminalConnection(new TcpTerminalConnection.Builder()
                        .SocketAddress(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9000))
                        .ConnectionInitiationType(ConnectionInitiationType.EcrInitiated)
                        .ConnectionManagementType(ConnectionManagementType.ConnectionPerSession
)
                        .Build())
                    .Build();

                // Create terminal
                vim.CreateTerminal(terminalInformation);
            }
            catch (InvalidOperationException e)
            {
                // Invalid VIM state
            }
            catch (ArgumentException e)
            {
                // Invalid parameters provided
            }
        }

        public void OnTerminalConnect(TerminalConnectEvent connectEvent)
        {
            // Get payment terminal information
            // Accept payment terminal connecting to VIM
            connectEvent.AcceptTerminal();
        }

        public void OnTerminalReady(TerminalReadyEvent readyEvent)
        {
            // Get VimApi instance for interacting with the payment terminal
            Terminal terminal = readyEvent.Terminal;
            terminalApi = terminal.GetVimApi(this);
        }

        public void OnConnectionChangedEvent(ConnectionChangedEvent connectionChangedEvent)
        {
            // Handle connection change events
        }

        public void OnCommunicationErrorEvent(CommunicationErrorEvent communicationErrorEvent)
        {
            // Handle communication error events
        }

        public void OnCardEvent(CardEvent cardEvent)
        {
            // Handle card events
        }

        public void OnTerminalMaintenanceEvent(MaintenanceEvent maintenanceEvent)
        {
            // Handle terminal maintenance events
        }
        public void OnBarcodeScanEvent(BarcodeScanEvent barcodeScanEvent)
        {
            // Handle barcode events
        }

        public void OnAccountSearchEvent(AccountSearchEvent accountSearchEvent)
        {
            // Handle account search events
        }


        public void OnVimError(VimErrorEvent evnt)
        {
            // Error handling
        }
    }
}
```

10.3 MixedConnectMode

When using *MixedConnectMode* VIM will both accept connections from payment terminals and allow the ECR software to create terminals where the connection is initiated from the ECR side.

This is only relevant if multiple terminals connect to the same VIM instance, and the termianls use different connection modes. A payment terminal can only be configured to use one of the two connection modes.

11 Appendix C: .net dependencies

The .net version of VIM have the following mandatory dependencies:

Library	Description	Link
Newtonsoft.Json	json library	https://www.newtonsoft.com/json
NLog	logging framework	https://nlog-project.org

All dependencies are available from https://www.nuget.org and will typically be automatically installed when VIM.net is installed with the NuGet package manager.