# UniTerm® Payment Interface Application

## UniTerm Integration and Deployment Guide

Revision: 8.3.0
Publication date March 06, 2017

# UniTerm Integration and Deployment Guide

Main Street Softworks, Inc.

Revision: 8.3.0

Publication date March 06, 2017


## Legal Notice

# Table of Contents

# 1 Revision History

| Version | Date | Changes |
|---|---|---|
| v8.3.0 | 2017-03-06 | • New First Data certification for EMV Debit and EMV Contactless<br>• Update recommended RBA versions.<br>• Add new cdcvm card holder verification method response. |
| v8.2.5 | 2017-01-19 | • Additional clarifications for use of the iOS framework.<br>• Add DELAYRESPONSE u_flags.<br>• Add fullscreen modifier to guimode configuration. |
| v8.2.4 | 2016-12-21 | • Clarifications for use of the iOS framework.<br>• Various other clarifications and fixes for typos.<br>• Addition of u_flowflags response parameter. |
| v8.2.3 | 2016-12-01 | • Add serialquirks flags for uniterm.ini |
| v8.2.2 | 2016-11-15 | • Add u_foodamount value of maybe for txnstart.<br>• u_cardclass=EBT has been split into EBTFS and EBTCB for clarity.<br>• Add [uniterm] configuration parameter returnbin to return the first 6 digits (in addition to the last 4 digits) of a card number in the txnstart response message so transaction decisions can be made (e.g. health benefits qualification). |
| v8.2.0 | 2016-10-17 | • All references to monetra_uniterm have been changed to just uniterm, including executable names and paths.<br>• Added EMV industry question<br>• Split out u_action values and descriptions into their own section for clarity.<br>• GUI can now use Device Licensing when passing a u_flags or GUIONLY while also passing valid values for device and devicetype. This prevents requiring duplicate licensing for workstations with physical devices that sometimes perform gui card entry.<br>• Ingenico RBA recommended version is now 18.04<br>• Linux HID device permission section has been added<br>• Asynchronous processing (multiplexing/interleaving transactions on a single connection) is now supported.<br>• QuickChip overview.<br>• USB is now known as HID, all usb references changed, hidlist report parameters and u_device format have also changed<br>• New commands (u_action values):<br>  • txnstart<br>  • txnfinish<br>  • deviceupload<br>  • deviceprint<br>  • devicereboot |

| Version | Date | Changes |
|---|---|---|
| | | • `reqsignature`<br>• `reqconfirm`<br>• `reqinput`<br><br>Associated request parameters for these new commands:<br>• `u_b64data`<br>• `u_filename`<br>• `u_text`<br>• `u_message`<br>• `u_inputtype`<br><br>Associated response parameters for these new commands:<br>• `u_needreboot`<br>• `u_confirmed`<br>• `u_input`<br>• `u_signature`<br>• Interchange requirement information.<br>• New/Changed configuration parameters:<br>• `[monetra] persist_conn` - Default changed to "no"<br>• `[uniterm] tippercent`<br>• `[uniterm] cashbackamount`<br>• `[uniterm] cashbackmax`<br>• Note that `uniterm.ini` isn't actually distributed anymore, instead a template of `uniterm_example.ini` is provided.<br>• New `u_flags`:<br>• `NOTIP`<br>• `NOCASHBACK`<br>• `NOCONFIRM`<br>• `NOSIGNATURE`<br>• Section added on Tip processing, including the addition of the `u_tip` response parameter.<br>• Section added on Cash Back processing, including the addition of the `u_cashback` response parameter.<br>• Section added on EBT processing, including the addition of the `u_foodamount` request parameter and `u_wasfood` response parameter.<br>• The OS-provided SSL/TLS root trust list is now automatically loaded and there is no longer an included `caroots` directory.<br>• In Deployment, discuss the new standalone installers provided.<br>• Update `devicetypes` functionality values.<br>• Update Apple iOS integration information for the framework variant.<br>• New `Ingenico RBA` form of `UTASEL.K3Z` used.<br>• Add guidance for Pay at the Table.<br>• Add information about automatic signature capture.<br>• Note that `CPX/uCPX` is limited to UniTerm v8.0, and not supported on v8.2 |

| Version | Date | Changes |
|---|---|---|
| | | • Configuration parameter `nosigfloor` has been changed to `nocvmfloor`<br>• `modifyconfig` parameters now use "`.`" as a delimiter instead of "`/`"<br>• `u_action=status` request now returns a `u_status` machine-readable status code<br>• Add section on updating `RBA` firmware.<br>• Add `skipped rcpt_emv_cvm` return value |
| v8.0.3 | 2016-03-02 | • Note limitation for asynchronous processing (or lack thereof)<br>• Update status of various certifications<br>• Add sections relating to iOS support<br>• Updated device application version support |
| v8.0.2 | 2015-11-16 | • Update status of various certifications<br>• Add section on obtaining devices.<br>• `UID_NOT_FOUND` error code addition, and clarify possible `u_errorcode` values for status requests.<br>• Missing required permission of `CARDTYPE` added.<br>• Added `persist_conn` configuration parameter for the `[monetra]` section.<br>• Added `deviceinfo` transaction type to request information about the connected device.<br>• Added `GIFTPIN` `u_flags` to prompt for PIN entry if a gift card is used. |
| v8.0.1 | 2015-09-28 | • Re-word and clarify configuration information for `Ingenico RBA`<br>• Clarify fields can be sent in to pre-populate key entry fields in GUI mode.<br>• Receipt should show "CALL ISSUER" for a response code of "CALL".<br>• Cardholder Name on receipts can be printed under the signature line.<br>• Document how to configure an `RBA` device for contactless.<br>• Remove unused `monetra_host` and `monetra_port` variables in example code.<br>• Add touchscreen mode support via `[uniterm]` config option of `guimode=touchscreen`.<br>• SSL is no longer a valid communication option for Android. Only Service communication is allowed.<br>• Add `ssl_auth_key` and `ssl_auth_cert` configuration parameters for the `[monetra]` section.<br>• Add `ssl_cert_validate` configuration parameter for the `[monetra]` section.<br>• Add `ssl_cadir` configuration parameter for the `[monetra]` section.<br>• `Verifone` recommended `XPI` version updated to 8.24D.<br>• `Ingenico` recommended `RBA` version updated to 15.06. |

| Version | Date | Changes |
|---|---|---|
| | | • `Ingenico RBA` added support for `USB-HID`.<br>• Update receipt examples to the latest generated by UniTerm Tester. |
| v8.0.0 | 2015-08-17 | • Initial revision |

# 2 UniTerm System

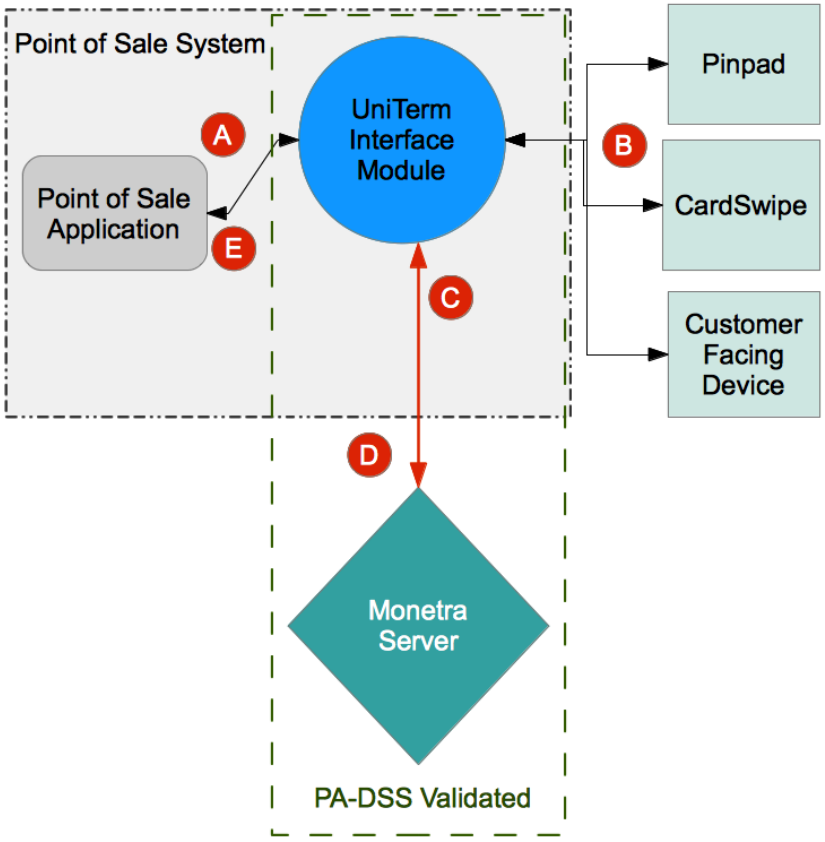


## 2.1 Overview

UniTerm securely handles sensitive cardholder data independent of the merchants application software. In addition, UniTerm provides a simple consistent interface to multiple payment acceptance devices such as card readers, pinpads and terminals.

## 2.2 UniTerm Architecture

The UniTerm module is accessed via its 'Transaction Request' mode, as described below:




| | |
|---|---|
| A | Point of sale application calls UniTerm for `txnrequest` (such as a sale transaction request) and includes basic information such as the amount of the sale and an order-number. |
| B | UniTerm communicates with devices (such as pinpads and card readers) to retrieve sensitive data, depending on request type (step A). |

| | |
|---|---|
| C | UniTerm sends the full transaction data-set to the Monetra server for further processing. |
| D | The Monetra server processes the transaction request (such as a sale) against the appropriate end point (for example TSYS) and then sends back the response it receives to the UniTerm module. |
| E | The UniTerm module then returns the transaction response back to the calling application. |

## 2.3 Design Decisions

UniTerm is designed to run as an independent application running in a separate address space from any integrated applications. The design decisions behind this are due to the PCI PA-DSS and EMV certification requirements, where a clear line can be drawn between the certified application (UniTerm) and the POS without any ambiguity as to if the POS would fall into scope. If this clear line was not drawn, the POS may be required to undergo the extremely intensive and costly EMV Brand Certifications, not to mention fall into scope for PCI PA-DSS.

Note: Due to the nature of iOS, UniTerm is also available as a library under special exception from Visa, which has stated they will not require the POS vendor on iOS to undergo EMV Brand or PCI PA-DSS validations when using UniTerm.

# 3 UniTerm Integration and Deployment Overview



## 3.1 Deployment

UniTerm is provided to developers during the integration process as either an OS-specific standalone installer, or as an installable license delivered via the Monetra Installer.

For production deployments on platforms other than Android and iOS, UniTerm should be bundled and distributed with the POS system under a distribution license from Main Street Softworks, Inc. Integrators should package the UniTerm directory that is created after UniTerm installation, and distribute that directory packaged together with their own components (e.g. POS application software). The UniTerm directory is self-contained and can be relocated to any path the integrator sees fit, without any additional system dependencies as long as the paths for any sub-directories included with the UniTerm installation (if applicable) are kept in the same relative paths in relation to the UniTerm executable.

It is also acceptable to redistribute the standalone installation package, however, the Monetra Installer method should never be used for production deployments.

For Android deployments, the UniTerm app is available via the Google Play Store, and it may be installed from there without a distribution license. Alternatively, by obtaining a distribution license, the UniTerm software can be provided as a standalone APK package for Android

ARM devices running v4.1.2 or later, or it can be bundled with the application software as a Service component.

For Apple iOS deployments, there are two distribution methods. The first is a framework that can be embedded into an application. Note that for iOS apps using this framework to be accepted into the App Store, Apple as well as the device manufacturers must provide approval for each type of BlueTooth device that will be used with the application. The second distribution method involves installation of the UniTerm application from the iOS App Store and using URL schemes for Inter-Process communication. The UniTerm App has not yet been approved by Apple due to Apple's requirements for each application to have a complete UI meant to be used outside of any features such as URL Schemes, the application is expected to be complete in Q1 2017, however a beta version can be provided upon request. Those interested in the framework for iOS should contact their sales representative.

Please contact sales@monetra.com for guidance on distribution licensing and bundling.

## 3.2 Versioning

### 3.2.1 Version Scheme

The versioning scheme employed by UniTerm is formatted as `X.Y.Z`, where each `X`, `Y`, and `Z` components are numeric-only version indicators separated by a period. Each numeric component may be from one to three digits in length. All software distribution updates will result in at least one of the components being updated.

The `X` component of the version indicates the product major version number. The major version component only changes when there are significant feature changes, or the changes impact any part of a security standard, such as PCI PA-DSS.

The `Y` component of the version indicates a product minor version change. The minor version will change when there are minor feature enhancements that do not impact the part of any security standard such as PCI PA-DSS.

The `Z` component of the version indicates a bug-fix release. Bug-fix releases do not change the overall feature-set or functionality of UniTerm, but may include security related fixes such as updates to 3rd party libraries (e.g. cryptographic libraries) distributed with UniTerm.

### 3.2.2 Wildcard Versioning

PCI PA-DSS requires a specific wildcard versioning definition which corresponds to the release which is being validated for compliance. With this release of UniTerm, the official wildcard versioning is `8.Y.Z`. The major (`X`) version number component is fixed at `8`, which as per the versioning definition states there will be no major feature changes or changes which impact the PCI PA-DSS standard (e.g. all changes that do not affect the major version number are classified as "no impact" changes). The minor (`Y`) and bug-fix (`Z`) wildcard components comply with the descriptions in the previous section.

Any future change which results in a change to the major version number will have a corresponding PCI PA-DSS validation.

## 3.3 Licensing

All UniTerm licensing is managed at the server level by the Monetra system with which UniTerm is connected. Since licensing is administered at the server level, there is nothing unique that needs to be deployed with UniTerm on the client side (such as a license or certificate file).

### 3.3.1 Registration

UniTerm generates unique ids for each connected `device` in order to send to Monetra to track the number of UniTerm licenses in use.

When UniTerm is started, during the first transaction and every 24hrs thereafter, the unique device ID will be automatically registered with Monetra. If this device is already associated with a UniTerm license, the license meta-data will be updated. If the device is not currently associated with a UniTerm license, Monetra will register this unique device id if a UniTerm license slot is available, otherwise Monetra will reject the registration request and UniTerm will cancel the transaction.

### 3.3.2 Device Definition

A device is either a physical Point of Interaction device, or a Graphical User Interface of the computer in which UniTerm is running.

Each physical device will consume a UniTerm license, the license is tied to the device serial number. Since the license is tied to the device, the physical device may be transferred to different POS stations without consuming additional licensing.

The use of the GUI mode in UniTerm, whether used with keyboard emulation card readers, for acceptance of manually keyed card entry, will also consume a UniTerm license. This license will be generated based on the unique machine ID or the MAC address of the first NIC. However if a physical customer-facing card entry device is also present, passing the `device` and `devicetype` parameters along with a `u_flags` of `GUIONLY` will use the device's licensing rather than registering the workstation itself, thus saving on duplicate licensing for workstations that use both a physical device and GUI modes of operation.

Note: If a physical device is being used and GUI mode is also used, if care is not taken, this will consume 2 UniTerm licenses, one for the physical device and one for the GUI mode of operation. Please see the `u_flags` of `GUIONLY`.

### 3.3.3 Management

Since UniTerm licensing is managed at the Monetra server, all license administration (view licenses, delete licenses, etc) can be performed using either the Monetra Administrator GUI or via the Monetra API. To more easily help identify and manage licenses, additional data is available in the license list such as: initial creation timestamp, last used timestamp, last used username, device type, and device serial number.

Note: If a UniTerm license (device or GUI) is removed (de-registered) from Monetra, then the license slot is not eligible to be re-used for 7 days. However, if the same [deleted] device is re-presented, it can immediately re-consume the license slot.

## 3.4 Starting UniTerm

For Desktop based deployments, the UniTerm module must be launched by the POS application software and should not be started at startup. If the POS system does not start UniTerm, then it is possible UniTerm will not be able to obtain screen focus for on-screen prompts.

For Android deployments, UniTerm should be automatically started at Boot, and simply Binding to the already-running service is sufficient.

For iOS deployments, the application is bound to a URL Scheme during installation and will automatically load when the URL Scheme is called.

### 3.4.1 Command Line Options

When starting UniTerm for Desktop based deployments, there are a few command line options supported that control the behavior.

- `-c` - Full path to the `ini` file to read. If not specified, it searches for the `uniterm.ini` in the paths documented in Section 3.7.
- `-p` - Port for UniTerm to listen on for incoming connections. If not specified, the value in the `ini` file is used. The purpose of this configuration value is to aid in the ability to start multiple UniTerm instances on the same machine with the intention of using GUI mode for multiple user logins (e.g. Terminal Services).
- `-h` - Help options are displayed.

## 3.5 Multiple Instances

When running UniTerm in conjunction with `Citrix` or Terminal Services, with the intention of using GUI mode, it is necessary to start multiple instances of UniTerm on the same machine. This can be accomplished by using a different port for each UniTerm instance. The port can either be configured via the command line options or by specifying a different UniTerm `ini` file. The integrated application would then communicate with UniTerm on its own dedicated port to prevent interference with any other UniTerm instances. The dedicated UniTerm instance must still be started by the POS application in that user instance otherwise UniTerm will be unable to display information or prompts.

If using UniTerm in device-only mode, it is recommended to use only a single instance of UniTerm and not start multiple instances. UniTerm is designed to be able to handle multiple transactions across multiple devices without the need for additional instances.

## 3.6 Swapping Devices

From time to time it may be necessary to swap out devices, whether the device is malfunctioning, being updated to a new firmware load, or simply being relocated. When

a device is swapped, UniTerm needs to be made aware of this, otherwise there could be unexpected behavior. In order to reduce transaction latency as much as possible, the first time a device is used after a fresh UniTerm start, UniTerm performs many queries against the device which may take many seconds to complete. These queries gather device information such as its type and capabilities and ensure the proper configuration parameters are loaded. In extreme cases this first transaction may detect a full device load is necessary which could extend this time to many minutes and result in a device reboot. On all subsequent transactions, these initial steps are stored in an in-memory cache and will not be repeated unless UniTerm is explicitly told to do so. When a device is swapped out, UniTerm may have no way to know this has occurred since it is operating on this cached data.

In order to tell UniTerm that a device has been swapped out, simply send a `u_action=deviceload` request or restart UniTerm. Either of these actions will force UniTerm to clear its in-memory cache and connect to the device as if it was the first transaction.

In some cases if the device itself isn't swapped (so the serial number has not changed), but instead the device has been manually cleared, such as when performing a firmware update, additional steps may need to be taken to ensure EMV parameters are loaded. There may be no way for UniTerm to determine if the device has the latest EMV parameters so UniTerm caches the `loadid` associated with the device serial number in the `uniterm.ini`. If this on-disk cache is incorrect because the device was manipulated outside of UniTerm, UniTerm must be informed of this by passing `u_forceload=yes` with the `u_action=deviceload` request. The `u_forceload` will tell UniTerm to ignore the `loadid` cache forcibly loading the EMV parameters into the device. In fact, it may be prudent to explicitly use `u_forceload` any time a device is swapped to ensure all data is loaded into the device.

## 3.7 Configuration Files

There is a single configuration file named `uniterm.ini` that must be configured before UniTerm can be used. Included with UniTerm is a file named `uniterm_example.ini` that can be used as a template.

The `uniterm.ini` file MUST be readable and writable by the UniTerm process.

### 3.7.1 Location

The location of the `uniterm.ini` file may vary from system to system, and the default search paths, listed in priority order, are:

- Windows:
  - `%APPDATA%/UniTerm/uniterm.ini`
  - same path as the `uniterm.exe` executable
- Mac OS X:
  - `~/Library/Application Support/UniTerm/uniterm.ini`
- Linux/Unix:
  - `~/.uniterm/uniterm.ini`

- same path as the `uniterm` executable
- iOS/Android: N/A - embedded into the application bundle, meant to be modified via GUI or `u_action=modifyconfig`

Note: If the `uniterm.ini` file cannot be located, or does not have proper read and write access, UniTerm will still start listening on the default port 8123 and return an `INI` related `u_errorcode` on all requests with a description of the issue. It should be noted that once the error has been corrected, UniTerm must be restarted to clear the error condition to force UniTerm to re-read its `INI` file.

### 3.7.2 Parameters

The parameters in this section are in standard `ini` format grouped by sections. Sections are in the format of "`[section]`". The settings for each section are in key/value pair format of "`key=value`". Each setting and section are delimited from each other using a new line, either Unix `LF` or Windows-style `CRLF` are acceptable.

Note: If editing the example `ini` file on Windows, using `Notepad.exe` is not advised. The example file uses Unix-style line endings that `Notepad.exe` cannot understand so it will make the entire file appear as a single line or otherwise not formatted correctly. It is known that `Wordpad.exe` does in fact understand Unix line endings and can be used successfully.

#### 3.7.2.1 Section: `[monetra]`

- `host`: Required. Hostname/address where Monetra resides
- `port`: Required. Port to connect to Monetra on
- `ssl_cert_validate`: Optional. Controls validation of the Monetra SSL server certificate. Possible Values are:
  - `full` - validate server certificate signature and require the full domain matches the certificate
  - `fuzzy` - validate server certificate signature and only the base domain matches the certificate
  - `validate` - validate only the server certificate signature, do not validate the domain name in the certificate
  - `none` - perform no server certificate validation

  If no value is specified, defaults to `none`. Monetra, by default, deploys a self-signed SSL certificate. You must either explicitly deploy a signed certificate (signed by a trusted CA) with Monetra to be able to turn on certificate validation, or add Monetra's self-signed SSL certificate to the trust list (see `ssl_cadir`).
- `ssl_cadir`: Optional. Path to a directory containing a list of the `PEM`-encoded trusted SSL Certificate Authority roots or individual server certificates to be added to the trust list. If available, the OS-provided trust list will be loaded first and any certificates in the provided directory will be appended. Only used when `ssl_cert_validate` is set to a value other than `none`.

- `ssl_auth_key`: Optional. Path to the SSL client certificate key used for two factor authentication. If not specified, the server will not be able to validate the authenticity of the client, however most deployments will not utilize this level of verification. Must be specified if `ssl_auth_cert` is specified.

- `ssl_auth_cert`: Optional. Path to the SSL client certificate used for two factor authentication. If not specified, the server will not be able to validate the authenticity of the client, however most deployments will not utilize this level of verification. Must be specified if `ssl_auth_key` is specified.

- `persist_conn`: Optional. Whether or not UniTerm should maintain persistent connections to Monetra, or if it should disconnect when there are no active transactions. If not specified, defaults to no.

#### 3.7.2.2 Section: `[uniterm]`

##### 3.7.2.2.1 Operating Parameters

- `port`: Required on all except Android. Port to listen on for incoming connections.

- `sharedsecret`: Optional. The value specified is the shared secret to use for the communication protocol. A value must be set if one wishes to allow remote connections (along with `localonly=no`), or to enable the `MODIFYCONFIG` command. When this configuration parameter is set, all requests to UniTerm must include the `u_sharedsecret` protocol-level key/value pair set to the same value.

- `localonly`: Optional. If not specified, defaults to yes. If set to no, a `sharedsecret` must also be set and remote connections will be allowed.

- `ssl_cert`: Optional. If not specified attempts to locate `ssl.crt` in the same path as `uniterm.ini`

- `ssl_key`: Optional. If not specified attempts to locate `ssl.key` in the same path as `uniterm.ini`

- `serialquirks`: Optional. Controls how serial connections are handled, some platforms may need special settings to operate reliably, but most systems should not configure these flags. This setting is controlled by a set of pipe (|) delimited flags as listed below:
  - `ignore_termios_failure` - Ignore errors while setting communications settings. This may be necessary on certain types of serial port emulators that do not allow this.
  - `no_flush_on_close` - Do not flush the serial port buffers on close.
  - `no_restore_on_close` - Do not restore the original configuration for the serial port on close.
  - `async_timeout` - When using asynchronous reads, allow the read operation to timeout rather than continue indefinitely, some serial port emulators may lock up without this flag. This flag is ignored when used with busy_polling. Windows Only.
  - `busy_polling` - Perform busy polling in a separate thread, rather than using asynchronous reads. This may be necessary for some serial port emulators that do not properly support Overlapped operations. Windows Only.

#### 3.7.2.2.2 Feature Parameters

- `idle_message`: Optional. Set the default idle message displayed on any device when not processing a transaction. This can be overwritten on a per-device level using the `u_deviceidlemessage` parameter in the protocol. This is not supported on all devices.
- `unsupportedcard`: Optional. If not specified, defaults to no. If set to yes, this allows trackdata to be returned to the caller for `txnrequest` and `cardrequest` only when the card type is confirmed to be non-financial. This is to allow in-store private-label gift (on `txnrequest`) as well as manager cards. The card must be returned unencrypted from the reader to be supported.
- `nocvmfloor`: Optional. If not specified, defaults to disabled, should be specified as a dollar amount. This configuration value will disable cardholder verification (e.g. PIN or Signature) when the transaction amount is less than this limit. For instance if the value is set to 50.00, and a 40.00 authorization is attempted as a swipe transaction, they will NOT be prompted to sign, however a 60.00 authorization would be prompted to sign. This feature works with EMV contact as well, however EMV Contactless has its own set of limits advertised by Monetra that UniTerm will honor. Applies only to Credit Card purchases.
- `guimode`: Valid options are `normal` and `touchscreen`. A modifier value of `fullscreen` can also be added to force the GUI to be rendered full screen rather than in a window, the modifier will be separated from the mode by a pipe (`|`), for example `touchscreen|fullscreen`. If no `guimode` is specified, it defaults to `normal`. `TouchScreen` mode enlarges all text in dialogs and provides an on-screen keypad to be used for manual card entry. Only numeric input is allowed in `touchscreen` mode, if an alpha-numeric Postal code needs to be entered (such as for Canada), a keyboard must be used. `Normal` mode is designed for use at a workstation with a keyboard and mouse.
- `tippercent`: Pipe-delimited (|) field containing whole percentage options for choices to be presented to the cardholder, or the keyword "`Other`" to allow a cardholder to enter a custom amount.

  If this setting is configured, it will enable tip prompting unless the `NOTIP` `u_flags` parameter is passed with the transaction.

  Though there is no limit to the number of values that may be requested, the options on the Right will be truncated (except for the `Other` option) if the device is not capable of presenting the number requested. It is therefore recommended that the lowest common denominator of 3 values be presented.

  Example: `tippercent=18|20|Other`
- `cashbackamount`: Pipe-delimited (|) field containing whole dollar amount options for choices to be presented to the cardholder, or the keyword "`Other`" to allow a cardholder to enter a custom amount. The maximum dollar amount allowed can be controlled by the `cashbackmax` configuration value.

  If this setting is configured, it will enable cash back prompting on supported card types (Debit, EBT Cash Benefits) unless the `NOCASHBACK` `u_flags` parameter is passed with the transaction.

Though there is no limit to the number of values that may be requested, the options on the Right will be truncated (except for the `Other` option) if the device is not capable of presenting the number requested. It is therefore recommended that the lowest common denominator of 3 values be presented.

Example: `cashbackamount=25|50|Other`

- `cashbackmax`: Maximum cash back amount allowed to be requested. If not specified, defaults to 100.00.
- `returnbin`: Optional. Boolean, defaults to no. If set to yes, this allows UniTerm to return the first 6 digits of the card number as part of the response for `txnstart`. This data can then be used for `BIN` checking to determine additional card classification such as if it is a health benefits card or qualified Visa card for Debt Repayment.

## 3.8 Communication

The communication protocol for UniTerm is very similar to that of Monetra. At the heart of the protocol is a simple key/value pair message structure, very similar to the Monetra Client Interface Protocol Specification. In fact, some of these key/value pairs sent to UniTerm are simply passed-through to Monetra for processing.

When communicating with UniTerm, you use standard network communications in Desktop and Server environments (Windows, MacOSX, Linux). For Mobile applications, the communication method is specific to what the OS allows for Inter-Process communication.

### 3.8.1 Network Communication

UniTerm supports both raw SSL communication with key/value pair transport, or XML over HTTPS. The protocol being used is autodetected by UniTerm on the first message sent by the POS. The standard APIs used with Monetra are also able to be used with UniTerm as they simply facilitate the same key/value pair transport mechanisms as the raw protocols. For more information on the underlying communications protocols or APIs, please reference the communications documentation and API documentation for Monetra.

Normally, UniTerm listens on localhost on port 8123, and as of UniTerm v8.2, listens on both IPv4 and IPv6 if available. It is possible to make UniTerm accept connections from remote machines by configuring a '`sharedsecret=`' set to a desired value as well as '`localonly=no`' in the `uniterm.ini`. When a shared secret is configured, all requests to UniTerm must include the `u_sharedsecret` parameter set to this secret value in order to prevent malicious requests.

### 3.8.2 Android Service Communication

The Android Service communication option utilizes `AIDL` in order to transmit the key/value pairs for each request to the UniTerm Service. Please see our Android SDK available at https://www.monetra.com/developers for an example of how to utilize this communication option.

### 3.8.3 Apple iOS

There are 2 methods of communication for Apple iOS. The first is an embedded framework with all included dependencies that is linked directly into the iOS application. The framework is available under special distribution license, please contact your sales representative for more information. The second method is an official Apple iOS App Store application that can be installed free and uses URL Schemes for Inter-Process communication, this application is expected to be accepted in Q1 2017, beta versions can be made available upon request.

#### 3.8.3.1 Framework

UniTerm on iOS can be provided bundled as a framework which allows private access to UniTerm's functionality. When building an application, the UniTerm framework itself along with the distributed dependencies (`libmonetra`, `openssl`, `zlib`), as well as the system-provided `ExternalAccessory` Framework and `libresolv.tbd` library must be added to the "Linked Frameworks and Libraries" for your project (In `XCode` under `Project->General`). The relevant frameworks and dependencies and a complete and working integration example will be provided with the UniTerm iOS Framework distribution.

Note: Both iOS device and simulator builds are provided for integrators. It should be noted, however, that simulator builds do not support External Accessories, so can only be used against IP/Ethernet-enabled devices. It is therefore recommended that developers have both Bluetooth and IP/Ethernet enabled devices for the various phases of development available.

A bundle with auxiliary files that will be installed is also provided. This bundle, called `uniterm.bundle`, must be added to your project in `XCode` under `Project -> Build Phases -> Copy Bundle Resources`. Within the bundle is a default `uniterm.ini`; this should be edited to reflect the Monetra location that should be used by the app. The `ssl_cadir` parameter should NOT be modified. The `port` parameter in the Uniterm section can be ignored or removed. When used as part of an iOS app connecting to UniTerm using a network connection is not supported. When a new build of UniTerm is distributed with the app, if there are differences in the `uniterm.ini` the new `uniterm.ini` will be merged with the old one. This only applies to the build number of the UniTerm library and not the build number of the app itself.

The app itself must be configured with the "Wireless Accessory Configuration" capability. Also, the `Info.plist` must list all external accessories it will be used with, along with other `MFi` protocols the device(s) supported advertise to prevent the user from being prompted to search the app store with a list of other applications that support this device. The current list of protocols is configured via Add "Supported external accessory protocols" with the below protocols (`UISupportedExternalAccessoryProtocols`):

- `com.ingenico.easypayemv.spm-transaction` - Actual used protocol
- `com.ingenico.easypayemv.spm-networkaccess`
- `com.ingenico.easypayemv.spm-pppchannel`
- `com.ingenico.easypayemv.barcodereader`
- `com.ingenico.easypayemv.spm-configuration`
- `com.ingenico.easypayemv.spm-sppchannel`

- `com.ingenico.easypayemv.printer`

The "Actual used protocol(s)" listed above will be part of the `u_device` in the key/value pairs sent to UniTerm (prefixed with `bt:` for bluetooth, which is used for `MFi`).

Note: For iOS apps using this framework to be accepted into the App Store, Apple as well as the device manufacturers must provide approval for each type of BlueTooth device that will be used with the application.

#### 3.8.3.1.1 APIs

| Function | Description |
|---|---|
| `BOOL uniterm_initialize( void );` | Initialize UniTerm. This must be called at app startup and before any other UniTerm functions are called. It is recommended to put it in the `AppDelegate`'s `init` function. |
| `BOOL uniterm_modify_config( NSDictionary *req );` | Modify Configuration. The `uniterm.ini` file should not be edited directly by the app after initialization. Instead the `uniterm_modify_config` should be used. This ensures that any UniTerm cached network connections are properly updated. This function must be used because `uniterm_run_trans` is explicitly blocked from modifying the configuration. Direct modification of the `uniterm.ini` is possible before `uniterm_initialize` is called. Both `uniterm.ini` and `uniterm-orig.ini` need to be changed. Specifically `uniterm-orig.ini` should be a copy of the `uniterm.ini`. |
| `NSDictionary *uniterm_run_trans( NSDictionary *req );` | Main interface for running transactions with UniTerm. It accepts UniTerm key value pairs and runs the request. It returns a dictionary of the key value pair responses. This is a blocking function and should be used with `dispatch_async`. When the call completes it should notify the app that it is finished so the app can take appropriate action with the response. |

### 3.8.3.2 URL Schemes

Apple iOS communication relies on the use of URL Schemes for Inter-Process communication. Information on this communication method can be found at https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html.

When an application calls UniTerm's registered URL Scheme, the UniTerm application will present itself in the foreground and process the requested transaction(s). Data passed via the URL scheme is JSON-encoded, containing the key/value pairs that make up a transaction as per this documentation. Multiple requests may be passed in a single message. Part of the request message is a response URL scheme for delivery of the response to the integrated application.

Please see our iOS Demo source code available at https://www.monetra.com/developers for an example of how to utilize this communication option.

Please note that Apple does not support the concept of a background service or true inter-process communication in the same way as Android supports, therefore UniTerm must run in the foreground while processing transactions. If this is not acceptable, then please contact us about the Framework version of UniTerm for iOS.

#### 3.8.3.2.1 URL Scheme messaging format

The JSON format is an object with strings for the keys and values. If multiple requests are to be sent at once, each request object may be encapsulated in an array, with each member in the array being a single request. When using request stacking, it is required that a unique `u_id` be passed with each request, which will be returned in the response and must be used for matching, never rely on the response array being in the same order as the request. If not using request stacking, `u_id` is not a required parameter.

Example JSON data:

```
[
 {
  "u_action": "devicetypes",
  "u_id": "1"
 },
 {
  "u_action": "bluetoothlist",
  "u_id": "2"
 },
 {
  "u_action": "version",
  "u_id": "3"
 }
]
```

For URL Schemes, UniTerm uses the `x-callback-url` (http://x-callback-url.com/) specification. All requests should be sent to "`uniterm://x-callback-url/transaction`" and must specify the following parameters:

- `x-source`: Textual name of the calling app.
- `x-success`: Return URL for successful requests. Will be used when UniTerm was able to process a request. This will be returned even when there was an error with the transaction itself such as a decline.
- `x-error`: Return URL on critical errors (such as parse failures). Will be used when there is an error in parsing the request. Such as malformed JSON data.

- `request`: The JSON request data, url-encoded.

Example Request:

```
uniterm://x-callback-url/transaction?
 x-source=MyApp&
 x-error=myapp://error&
 x-success=myapp://result&
 request=${URLENCODED-JSONDATA}
```

Example Critical Error:

```
myapp://error?
 errorCode=${CODE}&
 errorMessage=${MESSAGE}
```

Example Successful Response:

```
myapp://result?
 response=${URLENCODED-JSONDATA}
```

## 3.9 Shutting Down UniTerm

UniTerm should only be shut down if it was started by the POS, and does not apply to Android systems. On Windows, a standard Window shutdown message may be sent, or on Unix a `SIGTERM` signal may be sent to the UniTerm process. Or universally UniTerm supports a shutdown message via its protocol.

## 3.10 User Setup Permissions and Requirements

UniTerm requires that sensitive data never be returned from Monetra in order to ensure that the integrated POS is removed from PCI PA-DSS scope. In order to ensure this, UniTerm only allows merchant sub-users to authenticate. Sub users are unique usernames that can be tied to a merchant user with their own individual password, but provided only a subset of the permissions allowed. These unique usernames, when passed to UniTerm, are prefixed with the username of the merchant user and delimited with a colon (`:`). (e.g. `merchuser:subuser`).

UniTerm requires the sub user have the `obscure sensitive information` flag set, or it will generate a failure.

UniTerm also requires these permissions to operate:

- `CHKPWD`
- `CARDTYPE`
- `SALE`
- `VOID`
- `REVERSAL`

- `TERMLOAD` - Required if supporting EMV, Canadian Interac Debit, or `TransArmor`
- `EMVCOMPLETE` - Required only if supporting EMV
- `INTERACMAC` - Required only if supporting Canadian Interac Debit
- `ADMIN:MERCHINFO` - Used for populating receipt metadata and determining merchant card brands and capabilities in use
- `ADMIN:GETPERMS` - Used for verifying account setup.
- `ADMIN:IMAGEADD` - Only required if device support signature capture.

More permissions may be required based on the POS operations supported. Please consult with your integration and development team for the features used.

Note: Sub users may be created using the Monetra Client GUI after logging in, under `Admin -> SubUserManager`

## 3.11 Linux OS device access permissions

### 3.11.1 HID devices

Most Linux distributions, by default, do not allow non-root users to access HID devices. Since it is not desirable to run a POS or UniTerm as root, some system changes are required to grant access to normal users. In general, the `udev` subsystem controls device enumeration, so some rules must be added to tell it what permissions to grant for enumerated HID devices.

Before a `udev` rule can be added, a system administrator must determine what group to grant privileges to HID devices. On RedHat based systems, the most suitable pre-existing group name is probably `input`. For Debian based systems, the most suitable pre-existing group name is probably `plugdev`. If no suitable groups are pre-existing, the system administrator should create one with an appropriate name.

Next a file named `/etc/udev/rules.d/99-hid.rules` should be created with contents similar to:

```
KERNEL=="hidraw*", SUBSYSTEM=="hidraw", MODE="0660", GROUP="$group"
```

Of course, replacing `$group` with the desired group name.

Finally, the system administrator should add the user that wishes to run UniTerm to the group, a command to do that might look like:

```
usermod -a -G $group $user
```

Of course, replacing `$group` and `$user` as appropriate. If already logged in as the user being modified, it is necessary to log out and back in for the group membership to be updated.

Once these steps are performed, UniTerm should now be able to run as a non-root user and access HID devices

### 3.11.2 Serial devices

If the user that runs UniTerm is unable to open a serial device, most likely it is simply a group permissions issue. Both RedHat based and Debian systems use the `dialout` group for serial port access. A system administrator might need to do further research in to what group may be used on their system.

In order to add your user to the appropriate group, a command to do that might look like:

```
usermod -a -G $group $user
```

Of course, replacing `$group` and `$user` as appropriate. If already logged in as the user being modified, it is necessary to log out and back in for the group membership to be updated.

If your distribution is not setting group membership on serial devices during enumeration, it may be necessary to add specific `udev` rules to allow this. Please see the previous section for an example.

# 4 UniTerm Protocol



## 4.1 Overview

Application software communicates with UniTerm via the UniTerm protocol (which is intentionally similar to the Monetra Client Interface Protocol as they are meant to coexist). The protocol description documents only the key/value pairs that make up a transaction. These key/value pairs can then be transported using one of the supported communications protocols such as the standard Monetra SSL/TLS, possibly using one of our provided APIs to make communication easier, or XML over HTTPS. Usage of our Monetra SSL/TLS protocol is recommended as it allows asynchronous processing (multiple simultaneous, interleaved transactions) via a single connection (as of v8.2). Mobile devices may have their own specific method of transport.

Note: When processing transactions (such as `u_action=txnstart` or `u_action=txnrequest`), UniTerm simply augments the requests made with cardholder data captured either via a device or on-screen, and passes them on to Monetra. A majority of the parameters to be sent are simply pass-through to Monetra and not interpreted by UniTerm in any way; Therefore this guide must be used in conjunction with the Monetra Client Interface Protocol Specification to come up with a complete list of parameters necessary to complete transactions.

## 4.2 UniTerm Request Parameters

The table below describes the parameters used within the UniTerm protocol.

| PARAMETER | VALUE |
|---|---|
| `username` | The Monetra username to authenticate as. For security reasons this should be a restricted subuser account. |
| `password` | The Monetra password associated with username. |
| `u_action` | See UniTerm Actions (`u_action`) for a complete description of each action. |

| | |
|---|---|
| | • `deviceload`<br>• `txnrequest`<br>• `txnstart`<br>• `txnfinish`<br>• `cardrequest`<br>• `passthrough`<br>• `cancel`<br>• `deviceupload`<br>• `deviceprint`<br>• `devicereboot`<br>• `reqsignature`<br>• `reqconfirm`<br>• `reqinput`<br>• `devicetypes`<br>• `status`<br>• `seriallist`<br>• `bluetoothlist`<br>• `hidlist`<br>• `shutdown`<br>• `modifyconfig`<br>• `version`<br>• `deviceinfo` |
| `u_flags` | `Txnrequest`, `Txnstart`, `Txnfinish` and `Cardrequest` only. Multiple flags may be sent per data ticket request. All flags are separated by a pipe (\|) symbol.<br><br>• `ENCRYPTEDONLY` - Permit encrypted reader data only. Not valid on `cardrequest`<br>• `DEVICEONLY` - Suppresses display of clerk facing prompting and feedback, also known as GUI mode. For instance, on a swipe request, no swipe dialog would be presented nor would the clerk be informed of the status for customer-facing device interaction.<br><br>  Note: Setting this mode will prevent keyboard emulation readers from working, it will also prevent the ability for accepting clerk keyed input via a keyboard. On Android, iOS, and UniTerm launched in console mode, this flag is automatically implied as they do not support a GUI mode of operation.<br><br>• `GUIONLY` - Suppresses use of a referenced physical device. This flag can be used to utilize the physical device's UniTerm license for a clerk-facing GUI request, without using the device itself for card input. Without this flag, the GUI would register an additional UniTerm license based on the machine's unique id. |

| | |
|---|---|
| | • KEY - Perform capture of manually keyed data. Not valid on cardrequest. If an account or expdate field is passed in on the request to UniTerm, the fields will be pre-populated in GUI mode.<br><br>  Note: If a manually keyed EBT card is to be entered for a Food Stamp (SNAP) transaction, u_cardclass with a value of EBTFS must also be provided along with a qualified u_foodamount in order to go through the EBT flow and PIN prompting.<br>• AVS - Request AVS data. Only allowed on keyed transactions. If a street or zip field is passed in on the request to UniTerm, the fields will be pre-populated in GUI mode.<br>• CVV - Request for CVV data. Only allowed on keyed transactions.<br>• GIFTPIN - Request a PIN for gift cards, when a gift card is presented. Does nothing if a different card type is presented.<br>• NOTIP - Disable tip prompting if enabled via [uniterm] tippercent configuration.<br>• NOCASHBACK - Disable cash back prompting if enabled via [uniterm] cashbackamount configuration.<br>• NOCONFIRM - When performing a QuickChip finalization via txnfinish do NOT prompt the cardholder to confirm the amount prior to running the authorization.<br>• NOSIGNATURE - If using a signature-capable device, but an integrator does not want to use the auto-signature-capture capabilities in the UniTerm flow, this flag will treat the device as if it is not capable of accepting a signature. The u_need_signature response flag will be set appropriately for the transaction requirements.<br>• DELAYRESPONSE - This flag will cause UniTerm to send the transaction response back to the caller after the device has been closed. The device can be used immediately for a subsequent transaction with this flag, otherwise device closing will happen in the background which will hold a lock on the device for at least 100ms, or possibly longer. |
| u_cardclass | Txnrequest and Txnstart only. Optional. The card class that is expected (this provides a hint much like flags) to enforce only this card classification be allowed. UniTerm will normally prompt for the card type from the cardholder if it can not be determined based on the card presented. This option can be useful if the POS has already determined the card type prior to requesting UniTerm to process a transactions. If a card is presented that does not match the card class, the transaction will be rejected. Supported classes are: |

| | |
|---|---|
| | • `CREDIT` - Credit card transaction<br>• `DEBIT` - DEBIT card transaction<br>• `EBTFS` - Electronic Benefits Food Stamps (SNAP) transaction<br>• `EBTCB` - Electronic Benefits Cash Benefits transaction<br>• `INTERAC` - Canadian Interac card transaction<br>• `GIFT` - Gift card transaction |
| `u_device` | Requests: `Txnrequest, Txnstart, Cardrequest, DeviceInfo, DeviceLoad, DeviceUpload, DevicePrint, DeviceReboot, ReqSignature, ReqConfirm, ReqInput`<br><br>This specifies the path of the card entry device. Required parameter unless performing a GUI-based action (such as manual keyed entry, or swiping via a keyboard emulation card reader). Required if `u_devicetype` is provided.<br><br>• `HID[:serialnum] - HID (such as USB-HID), takes an optional serial number`<br>• `SER:port - Serial`<br>• `BT:mac,[uuid] - Bluetooth`<br>• `IP:ipaddr:port - IP` |
| `u_devicetype` | Requests: `Txnrequest, Txnstart, Cardrequest, DeviceInfo, DeviceLoad, DeviceUpload, DevicePrint, DeviceReboot, ReqSignature, ReqConfirm, ReqInput`<br><br>The unique device type supported by UniTerm as found via a `devicelist` request. Required if `u_device` is provided. |
| `u_language` | `Txnrequest` and `Txnstart` only. Used to override terminal defaults for display of text prompts. Current choices are "en" or fr". |
| `u_currency` | `Txnrequest` and `Txnstart` only. The numeric ISO currency code for EMV transactions (e.g. 840 for USD, 124 for CAD). Defaults to the terminals configured currency. This parameter is used to override the terminals default currency. |
| `u_deviceidlemessage` | `Txnrequest`, `Txnstart`, or `deviceload` only. Sets a message that the terminal should display when idle, this will be persistently cached by UniTerm and associated with the device serial number. |
| `u_forceload` | `deviceload` only. Values allowed are `yes`, `no`, and `full`. If not specified defaults to `no`. A value of `yes` will force a reload of all EMV parameters even if UniTerm believes the device already has the latest set of parameters. A value of `full` will additionally force reloading of all other objects UniTerm maintains, including but not limited to, configuration values, forms, and images. |

| | |
|---|---|
| u_id | Txnrequest, Txnstart, TxnFinish, Cardrequest, Status, and Cancel only. A unique id (generated by the calling application) that identifies the transaction. This is used for checking the status or canceling the transaction. Without this id the transaction state cannot be queried. |
| amount | Txnrequest and TxnFinish only. Transaction amount. Required. |
| u_b64data | Deviceupload only. Base64-encoded file data to upload. Required. |
| u_filename | Deviceupload only. Filename of the file to write. Required. |
| u_text | Deviceprint only. ASCII pre-formatted text to print to receipt printer. Required. |
| u_message | Reqconfirm only. Textual request message to prompt user for confirmation. Required. |
| u_inputtype | Reqinput only. Type of user input being requested. Required.<br><br>Possible values:<br><br>• PHONENUM - Request user to input a phone number. This may be useful for loyalty card prompting.<br>• INVOICENUM - Request user to input an invoice/order/ticket number |
| u_foodamount | Txnrequest, Txnstart, or Txnfinish only. Dollar amount indicating amount of qualified food purchases for EBT Food Stamps (SNAP). Or during an Txnstart the food amount may not yet be known, but to enable EBT prompting, a special value of maybe may be passed, followed by the the real amount in Txnfinish. If the amount passed in Txnfinish is zero, but EBT Food Stamps was selected by the card holder, the transaction will be aborted. |
| u_sharedsecret | Required on all transactions if a shared secret is set in the ini file. |
| monetra.host | Modifyconfig only. Modify the [monetra] host configuration value. |
| monetra.port | Modifyconfig only. Modify the [monetra] port configuration value. |
| uniterm.idle_message | Modifyconfig only. Modify the [uniterm] idle_message configuration value. |

### 4.2.1 UniTerm Actions (u_action)

The table below contains the descriptions of each possible UniTerm Action. The AUTH column indicates if the request requires authentication (username/password) or not.

| u_action | AUTH? | description |
|---|---|---|
| deviceload | Yes | Load a device with EMV and/or Interac parameters. This request will start a terminal download of EMV and/or Interac parameters to load into the device being used. Requires username, password, u_device, and u_devicetype parameters.<br><br>If the load for the device is identical to the previous load, the load will be skipped.<br><br>Please note this process may take up to 3 or 4 minutes depending on the processing institution being used and the device being used. The Device may also reboot during this process. It is strongly recommended to call this function when a lane opens, however if it is not called, it will automatically be performed prior to the first transaction.<br><br>If the device or merchant account does not support EMV or Interac, this command will simply return success.<br><br>Optionally u_deviceidlemessage may be passed to this as well to set the device's default message if supported by the device. |
| txnrequest | Yes | Full Transaction Request for card holder data. Request a transaction be performed, all normal Monetra transaction parameters should be included in the request that may be required for processing or to comply with interchange requirements (e.g. action, username, password, amount, nsf, ordernum, etc). Sensitive data, explicitly, should not be sent (e.g. trackdata, account, cvv2, pin) as that data will be retrieved by UniTerm either via the GUI or via a card entry device and forwarded to the Monetra server as part of the transaction request.<br><br>UniTerm will fully control the transaction flow and user prompting for such a transaction. This request should only be used when card data entry is required as part of the transaction, for other operations passthrough may be a more appropriate action. |
| txnstart | Yes | Transaction Start Request for card holder data. Request to start a QuickChip transaction, similar in function to a txnrequest message, but does not accept a transaction amount. Needs to be followed up with a txnfinish or cancel request. Once a card is presented and the cardholder has completed any necessary verification and removed their card, a response will be returned with card information metadata such as the card type, some basic EMV card parameters, and the last 4 digits of the card number. |

| | | |
|---|---|---|
| | | Though the purpose of this request is to support QuickChip, it still supports non-EMV transactions.<br><br>A `u_id` unique parameter is required to be sent with this request for completing the transaction.<br><br>See QuickChip for more information. |
| `txnfinish` | Yes | Transaction Finish Request. Request to finish a QuickChip transaction. Requires the integrator send the same `u_id` as was sent with the initial `txnstart` request. The integrator can choose to send additional parameters to pass through to Monetra with this request based on information returned from `txnstart` (e.g. card type). Will return the same response parameters as `txnrequest`.<br><br>See QuickChip for more information. |
| `cardrequest` | Yes | Non-Financial Card Entry Request. UniTerm will prompt for card entry, and if it is determined the card is non-financial, it will return the card data. This can be used for manager cards and private label gift cards that are not processed through Monetra. The card must be swiped, and the reader must be configured to return the card in unencrypted form. |
| `passthrough` | Yes | This action performs a direct pass-through of parameters to the Monetra server. Only the `username`, `password`, and `u_action` parameters are required. This method will simply proxy requests to the Monetra server and can be used for transaction modifications (e.g. voids) or for pulling reports from Monetra.<br><br>Do not use this request if cardholder data needs to be retrieved for the transaction.<br><br>Users may wish to send their requests directly to Monetra rather than using this feature. |
| `cancel` | Yes | Will attempt to cancel an outstanding `txnrequest` or `txnstart`. Requires `username`, `password` and `u_id` fields which must match the pending request. If the transaction cannot be canceled, such as if it is ineligible (such as when waiting on a response from the Monetra server), or the device doesn't support canceling the outstanding request, `u_errorcode` will return `PENDING_TRAN`. |
| `status` | Yes | Requests the current status of a pending `TXNREQUEST` or `txnstart/txnfinish`. Requires `username`, `password` and `u_id` fields which must match the pending request. This will provide a textual verbiage response suitable for clerk display. Will normally return a `u_errorcode` of `SUCCESS` when either the transaction is still in-progress or UniTerm is in a cleanup |

| | | |
|---|---|---|
| | | phase after a transaction. Will return a value of `UID_NOT_FOUND` if the request is no longer being processed. |
| `deviceupload` | Yes | Requests uploading the given file to the device at the requested device path. Not all devices support the concept of file uploads. Required parameters are `u_b64data, u_filename`, and may return `u_needreboot=yes` to indicate that a device reboot is required for the change to take effect (an automatic reboot will not occur so that additional files that may also require reboot can be uploaded). The `u_filename` should contain a properly formatted path for the device in use. Some devices like `Ingenico RBA` just want the filename with no path component. |
| `deviceprint` | Yes | Requests printing the given text to a built-in receipt printer if available. The text provided must be formatted properly for the given device. Only plain text receipts are supported. Receipt data is provided via the `u_text` parameter. |
| `devicereboot` | Yes | Requests the device to reboot. Typically this should only be done when an `u_needreboot=yes` parameter is returned in response to a file upload request. |
| `reqsignature` | Yes | Requests the device to prompt for signature capture for non-financial requests. This should only be used outside of a transaction flow (e.g `txnrequest` or `txnstart/txnfinish`). For financial transactions, a signature will be automatically captured and stored within Monetra so there is no need to call this function. The signature data will be directly returned to the caller in the `u_signature` response parameter.<br><br>Not supported in GUI mode |
| `reqconfirm` | Yes | Requests the user confirm a message presented on the screen. The message is supplied via the `u_message` request parameter, and the response is recorded in the `u_confirmed` response parameter.<br><br>Not supported in GUI mode |
| `reqinput` | Yes | Requests the user enter the input type specified via `u_inputtype`. The response data is returned in the `u_input` response parameter.<br><br>Supported in both GUI and device mode |
| `devicetypes` | No | Will return a comma separated list of device types supported by the UniTerm module. No authentication required.<br><br>Headers:<br><br>• `devicetype` - internal device name<br>• `manufacturer` - textual description of device manufacturer<br>• `model` - textual description of model |

| | | |
|---|---|---|
| | | • `connectivity` - pipe separated list of connectivity methods supported by the device, e.g. `SERIAL\|HID\|BLUETOOTH\|IP`<br>• `functionality` - pipe separated list of functionality supported by the device:<br><br>e.g. `EMV\|MAC\|E2E\|KEY\|SIGNATURE\|UPLOAD\|PRINT\|REBOOT\|CONFIRM\|FREEFORM` (`FreeForm` is used by `reqinput`) |
| `seriallist` | No | Will return a comma separated list of all serial ports enumerated on the system. No authentication required.<br><br>Headers:<br><br>• `port` - The port path<br>• `desc` - Description of port, if applicable |
| `bluetoothlist` | No | List all 'bluetooth' devices that have been paired with the machine UniTerm is running on. The device may or may not be present. Currently only supported on Android and iOS.<br><br>Headers:<br><br>• `name` - The textual name of the device as registered with the operating system.<br>• `mac` - The device bluetooth `MAC` address<br>• `uuid` - The device bluetooth `UUID` |
| `hidlist` | No | List all supported HID (possibly USB-HID) devices that are currently attached to the machine UniTerm is running on.<br><br>Headers:<br><br>• `devicetype` - The devicetype (device internal name) associated with the HID entry<br>• `manufacturer` - The manufacturer as advertised by the device.<br>• `model` - The pretty name for the device type as it is known to UniTerm<br>• `product` - The product as advertised by the device.<br>• `serialnum` - The serial number as advertised by the device. |
| `shutdown` | Yes | Terminates execution of the UniTerm. This should be called when the application software terminates. |
| `modifyconfig` | Yes | Allows a select number of `ini` configuration settings to be set via the API. In order to activate the ability to use this feature, an integrator must enable the shared secret in the configuration and the connection must come from the local machine. |
| `version` | No | Requests the current version of UniTerm. The version information is output in human-readable format in the verbiage response field. The version information also contains the build number. |

| | | |
|---|---|---|
| `deviceinfo` | Yes | Requests the information about the connected device. Returns response parameters of: `serialnum, device_manuf, device_model, device_app, device_appver, device_kernver` |

### 4.2.2 Interchange/Rate Qualification Requirements

UniTerm cannot generate all the parameters necessary for proper interchange qualification. An integrator must ensure they are passing all the required parameters for their industry, card brand, or special processing requirements. In general, there are a few parameters of greater importance than others, those that will apply to all merchants that integrators need to ensure they are aware of. For a complete list of parameters accepted, please reference the Monetra Client Interface Protocol Specification.

The table below contains a list of the most important parameters, please keep in mind this is not a complete list:

| Key | Required When | Description |
|---|---|---|
| `nsf` | Card Present | Required value: yes<br><br>Approve transactions even if there are insufficient funds (`NSF`). Will result in a partial approval if there are insufficient funds, an `authamount` response parameter will be returned for partial authorizations. When a partial authorization is received, a merchant should request another payment method for the remaining funds. It is acceptable to request a reversal if no additional payment methods are possible.<br><br>All card brands are requiring merchants support partial authorizations, merchants can be subjected to fines. Some issuers may return partial approvals even without this flag because of the requirements in place. |
| `laneid` | Card Present | The lane ID is a Mastercard requirement that each register or lane at a merchant's location be uniquely identified. This should be a numeric value no more than 8 digits in length, however some processors cannot support more than 2 or 4 digits. It should be sent on all transactions as it is not possible to know the card type prior to a request to UniTerm. |
| `ordernum` | Restaurant, Card Not Present, Level 2 | A merchant order number is required on all card not present (mail order and ecommerce) transactions, Restaurant, and all Level2 cards such as purchase or corporate cards. Since it is not possible to know the card type prior to a request to UniTerm an Order Number should be sent on all transactions. |

| | | |
|---|---|---|
| | | An order number is normally alpha-numeric up to 25 characters, however for restaurant a 6 digit order number should be used |
| examount | Restaurant | Extra Amount - Tip Amount. If a tip amount is known at the time of sale it must be populated, if using UniTerm's built-in tip-prompting, this field should not be passed as UniTerm will populate it. |
| zip | Key Entered | All key entered transactions should include at least a billing zipcode for best rate qualification. Often this is best accomplished by passing a u_flags of AVS, however there may be cases where the billing zipcode is already on file and will be presented separately. |

## 4.3 UniTerm Response Parameters

The UniTerm module will return all standard response tags from the Monetra server such as code=, cardtype=, and so on. The additional tags listed below are for transaction flow handling, please see the EMV Receipt section for additional tags that may be returned specific to receipt formatting.

| PARAMETER | VALUE |
|---|---|
| u_emv_chip_malfunction | (yes or not sent) = Indicates that there was a chip malfunction during EMV Complete. Note: Certain card brands require a note on the receipt stating there was a chip malfunction. |
| u_need_signature | (yes or not sent) = Monetra returns rcpt_emv_cvm which can have a value of "sig" saying signature is required. The u_need_signature means that a signature is required and it should be printed/obtained from the paper receipt. If an EMV requires a signature and one was not captured electronically, this flag indicates it should be obtained via a paper receipt. |
| u_errorcode | See section below. |
| u_cancelable | u_action=status only. Yes or No. Indicates if the current transaction state will allow a cancel to be sent. This is useful for showing and hiding a cancel button within an integration's GUI. |
| trackdata | u_action is txnrequest, txnstart, or cardrequest only. Also requires the ini configuration of unsupportedcard=yes. Will only be returned for non-financial cards that are returned unencrypted from the reader. Facilitates the use of manager cards as well as in-store private label gift systems that do not flow through Monetra. The u_errorcode returned with the response will always be NONFINANCIAL. Support during a txnrequest/txnstart is tailored to the use of |

| | |
|---|---|
| | private label gift cards and will only be returned when the cardholder selects `GIFT` from the payment type selection screen. |
| `serialnum` | `u_action=deviceinfo` only. Serial Number of connected device. |
| `device_manuf` | `u_action=deviceinfo` only. Device Manufacturer. |
| `device_model` | `u_action=deviceinfo` only. Device Model Name. |
| `device_app` | `u_action=deviceinfo` only. Application running on connected device. |
| `device_appver` | `u_action=deviceinfo` only. Version of the application running on the connected device. |
| `device_kernver` | `u_action=deviceinfo` only. EMV Level2 kernel version of the connected device. |
| `device_encryption` | `u_action=deviceinfo` only. Encryption type used by device, if any. |
| `u_needreboot` | `u_action=deviceupload` only. Will return "yes" when a device reboot is needed for the action performed to take effect. |
| `u_signature` | `u_action=reqsignature` only. Will return base64-encoded TIFF data representing the signature captured by the device. |
| `u_confirmed` | `u_action=reqconfirm` only. Will return "yes" or "no" depending on if the user confirmed the dialog. |
| `u_input` | `u_action=reqinput` only. Will return the user-entered data. |
| `u_tip` | `u_action` is `txnrequest` or `txnfinish`. Will return the tip amount chosen by the cardholder. |
| `u_cashback` | `u_action` is `txnrequest` or `txnfinish`. Will return the Cash Back amount chosen by the cardholder. |
| `u_wasfood` | `u_action` is `txnrequest` or `txnfinish`. Will return a boolean value indicating if the cardholder selected EBT Food Stamps (SNAP). |
| `u_status` | `u_action` is `status`. Machine readable status code. See UniTerm Status Codes. |
| `u_flowflags` | `u_action` is `txnrequest`, `txnstart`, or `txnfinish`. Pipe-delimited flags to provide integrators more insight into possible actions UniTerm has taken.<br><br>• `SIGCAPTURED` - Signature was captured<br>• `REVERSED` - The transaction was approved but then reversed. This is typically due to the customer canceling the transaction, or the card declined the transaction. |

## 4.4 UniTerm Error Codes

Errors will be returned in the `u_errorcode` field. Each error code may be used for more than one error type. Please see the verbiage response for more details. Note: On a successful transaction the `u_errorcode` will be set to SUCCESS but that only indicates communications with the Monetra server were successful. It does not mean the transaction was approved.

| u_errorcode | definition |
|---|---|
| MISSING_PARAM | A required parameter was missing. |
| INVALID_PARAM | A specified parameter was invalid |
| PENDING_TRAN | pending transaction already in progress |
| INVALID_USE | Typically means parameters specified should not have been specified together. |
| PERMISSION_ERROR | The user account within Monetra was misconfigured. |
| MONETRA_ERROR | There was an error communicating with Monetra. |
| DEVICE_ERROR | There was an error communicating with the card entry device. |
| CANCELED | User canceled request |
| FAILURE | Generic Failure |
| DEVICE_INUSE | The device specified is being used by another transaction. |
| UID_NOT_FOUND | A `u_id` was specified on a status or cancel request and no such `u_id` is actively being processed. |
| BAD_READ | The device returned a card read error. |
| MAC_FAILURE | The transaction was rejected because the MAC returned from the processor did not match the expected value. |
| EMV_CARD_DENY | The card locally declined the transaction. |
| EMV_CARD_REMOVED | The card was removed before the end of the transaction. |
| CARD_NOT_SUPPORTED | The card presented was not supported. |
| DEVICE_NOT_LOADED | The device needs to be loaded before it can run EMV transactions. |
| FALLBACK_NOTALLOWED | There was an error reading the chip and the card brand rule does not allow the card to be re-presented via another means. |
| INI_CANNOT_FIND | The `uniterm.ini` could not be found. |
| INI_CANNOT_READ | The `uniterm.ini` is not readable by the UniTerm process. |
| INI_CANNOT_WRITE | The `uniterm.ini` is not writable by the UniTerm process. |
| INI_INVALID_PARAM | The `uniterm.ini` has an invalid configuration parameter. |
| NONFINANCIAL | The card presented is not a financial card. This code will be returned when requesting and returning trackdata for non-financial cards when the configuration of `unsupportedcard=yes` is set. |

## 4.5 UniTerm Status Codes

Status codes returned via the `u_status` response parameter. Each status code may be a generalization used for more than one phase of the transaction flow. Please refer to the `verbiage` response parameter for a more descriptive human-readable status message.

| u_status | definition |
|---|---|
| WORKING | Generic status say UniTerm is processing. |
| CARD | Waiting for card presentation. With EMV card could be inserted and waiting for customer to choose language. |
| EMVFLOW | EMV processing. Typically waiting for customer to enter their PIN. |
| KEYED | Waiting for keyed account and related (zip, cvv). |
| PIN | Waiting for PIN entry. This may not be presented for EMV cards as the device may handle this internally. |
| CARDCLASS | Waiting for card class selection (Debit, EBT, Gift prompting) |
| TIP | Waiting for tip entry. |
| CASHBACK | Waiting for cash back entry. |
| SIGNATURE | Waiting for signature. |
| EMVCOMPLETE | EMV processing is completing. |
| CONFIRM | Waiting for user confirmation. |
| CONFIRMAMOUNT | Waiting for user to confirm the transaction amount. |
| CONFIRMFOOD | Waiting for user to confirm the food amount. |
| CONFIRMRETRY | Waiting for user to confirm to retry the transaction after a failure. |
| INPUT | Waiting on customer input (e.g. phone number). |
| MONETRA | Communication with Monetra. Typically waiting on a response from Monetra. |
| REMOVECARD | Waiting for card removal. |
| REVERSAL | Reversing transaction (typically due to card decline). |
| DEVICEPRINT | Printing receipt on device. |
| DEVICEOPEN | Waiting for device to open. |
| DEVICECLOSE | Waiting for device to close. |
| DEVICEREBOOT | Rebooting device. |
| CANCEL | Canceling transaction or operation. |

## 4.6 Tip Prompting

UniTerm supports prompting the cardholder for a tip amount at the time of payment when the `[uniterm] tippercent` configuration setting is configured and the `NOTIP` `u_flags` parameter is NOT provided.

When a customer has chosen to add a tip amount to a transaction, the `amount` provided by the POS to UniTerm will be incremented to reflect the tip amount and the `examount` will be populated with the tip amount when the transaction is sent to Monetra.

In the response returned by UniTerm, the tip amount will be provided to the POS in the `u_tip` response parameter.

Note: Special care should be taken to validate if the `authamount` response parameter is returned, indicating a partial authorization occurred, that split tender operations can occur. When prompting for a second payment method, an integrator should use the `NOTIP` `u_flags` in order to avoid tip prompting on the second method of payment, and respect the returned `u_tip` response for the chosen tip amount from the original response.

## 4.7 Cash Back Prompting

UniTerm supports prompting a cardholder if they would like to request Cash Back when presenting a Debit or EBT Cash Benefits card for payment. UniTerm will automatically perform such prompting if the `[uniterm] cashbackamount` configuration parameter is set and the `NOCASHBACK` `u_flags` parameter is NOT set. The `[uniterm] cashbackmax` configuration parameter can be used to limit the amount of Cash Back that can be requested.

When a customer has chosen to request Cash Back with a transaction, the `amount` provided by the POS to UniTerm will be incremented to reflect the Cash Back amount and the `cashbackamount` parameter will be populated with the Cash Back amount when the transaction is sent to Monetra.

In the response returned by UniTerm, the Cash Back amount will be provided to the POS in the `u_cashback` response parameter.

Note: Special care should be taken to validate if the `authamount` response parameter is returned, indicating a partial authorization occurred. If the returned amount is less than the requested `amount` plus `u_cashback` then the POS must decide on the proper course of action. For instance if the `authamount` is greater than requested, but less than the `amount` plus `u_cashback` then partial Cash Back would be provided to the cardholder. Otherwise if the `authamount` is less than the requested `amount`, then the `u_cashback` returned should be completely ignored and the POS would need to prompt the cardholder for another method of payment.

## 4.8 EBT Processing

UniTerm supports prompting if the card presented is an EBT card. If the `u_foodamount` parameter is populated with a non-zero dollar amount indicating the amount of the transaction that applies to qualified food purchases (or for `txnstart` with a value of `maybe`), then the customer will also be prompted if they would like to use Food Stamps (SNAP) or Cash Benefits to complete the transaction. When the cardholder makes the selection, UniTerm will internally rewrite the `action=sale` request parameter to `action=ebtfssale` or `action=ebtcbsale` as appropriate.

For `txnstart` transactions where `u_foodamount=maybe`, the integrator must send a valid `u_foodamount` value with `txnfinish` otherwise the transaction will be aborted.

If Food Stamps (SNAP) was selected, `u_wasfood` will be returned as yes/true to indicate this. If the requested `amount` is greater than `u_foodamount`, then a partial authorization will be returned (as indicated by the the `authamount` response parameter) indicating the amount of the authorization was less than the requested. This returned `authamount` may be less than the `u_foodamount` if there are insufficient funds, otherwise it will be equal to the `u_foodamount` requested.

If a partial authorization is performed, the merchant should perform a split-tender operation and prompt for another method of payment for the remainder, which may also be EBT. The requested `amount` and `u_foodamount` need to be adjusted accordingly on the next request based on the amounts previously authorized.

## 4.9 QuickChip

The card brands have coined the term QuickChip to refer to the modification of the EMV flow to reduce the amount of time a card must stay in the terminal, allowing the card to be presented as early as possible. The goal of QuickChip is to ensure no additional delay to the overall checkout process is incurred for EMV transactions when compared to an MSR transaction. It also is more similar to the existing MSR flow that customers may be acquainted with.

QuickChip is only available in the US Market, the card brands have not yet allowed this flow in other markets.

In order to use QuickChip with UniTerm, an integrator must use the `txnstart` and `txnfinish` messages rather than the `txnrequest` message. When the merchant opens the order to add items for purchase, they will immediately send the `txnstart` message, which will not contain an amount since the final amount is not yet known. Once the card holder has presented their card and completed any necessary cardholder verification, and removed their card, a response will be returned to the merchant. Once the final amount is known, the merchant will send a follow-up `txnfinish` message with the final amount and the transaction will go online for approval and the merchant will be returned all the receipt data just as if the request was a `txnrequest`.

If the amount is known at the time of the order, then the non-QuickChip `txnrequest` message should be used.

## 4.10 Pay at the Table

UniTerm provides all the tools needed for an integrator to support Pay at the Table. Though the method an integrator might choose can vary from system to system, we have outlined a suggested flow that should work for most environments when integrators choose to use a payment device specifically meant for pay at the table without necessitating the need for the use of an additional device such as a tablet. Those integrators that choose to use a tablet for pay at the table will not use the information contained in this section as the flow, in relation to UniTerm, would more closely resemble the standard Pay at the Counter flow.

The below flow assumes the customer will be entering their own invoice number, but a clerk may choose to perform that step on their behalf if there is only one ticket associated with the table.

- Print one or more detailed meal tickets for the table (split checks), and ensure a unique numeric order number for the day appears on the ticket. It is suggested these ticket numbers be 6 digits in length and randomly, not sequentially, generated to help prevent typos matching another open ticket. Deliver the tickets along with a payment device to the table, such as an `Ingenico IWL250` that is wireless and contains a built-in receipt printer.

- Send a `u_action=reqconfirm` request with a useful message/instruction to the user such as "`u_message=Enter ticket number from top of ticket on next screen.`", wait on a response. Ignore the actual response returned.

- Send a `u_action=reqinput` with `u_input=INVOICENUM` and wait on a response.

- Look up the requested Invoice, and assuming it is found and not already paid, request the user confirm it is the right ticket and amount using `u_action=reqconfirm`, when the user accepts the dialog, precede to the next step, if rejected, start over. If the invoice was not found, use `u_action=reqconfirm` to send back an error and restart the flow.

- Start the payment flow with `u_action=txnrequest` as you would in a normal clerk-attended environment. Tip prompting, etc will take place if UniTerm is configured to do so. On error or user cancellation, restart the flow, otherwise continue to the next step.

- If a signature is required, and the device does not support signature capture, format a receipt for the printer size on the device and send `u_action=deviceprint` with the `u_text` set to the plain text, pre-formatted, MERCHANT receipt data with a signature line.

- If a merchant receipt was printed, and the device being used does not have an automatic cutter, then you should prompt the user for how to tear the receipt then press enter to continue using the `u_action=reqconfirm`, and ignore the result of the request.

- Format a receipt for the printer size on the device and send `u_action=deviceprint` with the `u_text` set to the plain text, pre-formatted, CUSTOMER receipt data (no signature line).

- Start the flow over for more tickets.

## 4.11 Signature Capture

During a transaction, if UniTerm decides a signature is necessary to complete a transaction and the device is capable of capturing a signature, UniTerm will automatically prompt for signature and save it with the transaction. The signature will be kept on file for as long as the record of the transaction is kept on file. All signatures are stored in the `TIFF` image format within Monetra and may be retrieved via the `action=admin, admin=getimages` function call. Please see the "Protocol Addendum: Signature Capture/Storage" for Monetra for more information.

Note: Some devices, especially mobile devices, or web browsers may not natively support displaying the `TIFF` image format. It is recommended to use an open source library such as `ImageMagick` to assist in the conversion for display purposes.

# 5 EMV transactions with UniTerm



EMV transactions, by nature, are much more complex than traditional magnetic stripe transactions. UniTerm hides this complexity from the application software. In the case of magnetic stripe and EMV transaction, the application software will send the request to UniTerm. The device capabilities (EMV for example) will be determined by UniTerm, along with the merchant account configuration. From these UniTerm will handle the appropriate prompting and flow aspects related to the determined capabilities. The application software simply needs to send a `u_action=TXNREQUEST` and let UniTerm handle the rest.

## 5.1 Transaction Flow and Prompting

Integrators unfamiliar with EMV may notice some specific flow cases that seem counter-intuitive at first. This section is meant to address these EMV-specific cases.

### 5.1.1 Swipe prompts to insert

If a chip-enabled card is swiped on an EMV-capable terminal, it is mandated that the user be prompted to insert the card. This is an EMV certification requirement which cannot be lifted and it is meant to train consumers to insert their cards and to prevent fraud.

### 5.1.2 Tap prompts to insert

There are certain thresholds negotiated between the card and terminal which may request a chip-enabled card that is presented as a tap transaction be inserted instead. When this occurs, it can be due to a number of factors including fraud mitigation, or the card has determined it needs to be updated (for insert transactions, an issuer can choose to return issuer scripts to remotely reprogram cards).

### 5.1.3 Insert prompts to swipe

If a chip-enabled card is prompted to be swiped, this is usually an indication that there was a chip malfunction and the cardholder should have their card replaced, called a technical fallback. It is expected at some point in the future, technical fallback will be disallowed due to

fraud concerns. The other possibility is if the application id in use by the card is not supported by the terminal.

#### 5.1.4 PIN required on Credit Cards

The cardholder verification method is negotiated between the card and the terminal. If both the card and terminal support PIN entry, it may be chosen as the desired verification method. Consumers in the US may not expect to enter a PIN on their credit cards, but it is common among foreign cards.

#### 5.1.5 Signature not requested

The cardholder verification method is negotiated between the card and the terminal. They may negotiate Signature, PIN, or what is called `NoCVM` which means no cardholder verification is required for the transaction. The decision is strictly made based on the terminal capabilities and card capabilities.

#### 5.1.6 Tap transaction run as MSR on chip card, no insert requested

It is a requirement by the card brands that if a chip-capable card is presented as a tap that the card NOT be prompted for insertion. This can happen due to a terminal not being configured for contactless EMV support, or if a chip is malfunctioning.

#### 5.1.7 Immediate decline without contacting the processor

EMV cards have the ability to make decisions about the transaction before it is even processed. From time to time a merchant may see a chip card presented that results in an immediate decline before requesting cardholder verification or connecting to a processing institution. This could happen because the card has exceeded some internal threshold, or the card has received a remote script on a previous transaction to explicitly block transactions, such as a card block or application block.

### 5.2 Common questions

#### 5.2.1 How do I add a gratuity/tip to a transaction?

Tips are added to EMV authorizations just as they are with MSR authorizations, nothing has changed in the US rules. An integrator will simply send a `preauth` with the order amount, then when the tip amount is known, a `preauthcomplete` will be sent with the final order amount and `examount` will contain the tip amount. However, if the tip is greater than 20%, merchants should obtain a new authorization for the tip according to the card brand rules. Of course if the tip amount is known prior to the authorization, the tip amount should be included a part of the authorization request.

There is much confusion regarding tips in the US market with the introduction of EMV Chip and Pin, most of this is due to European rules which state the gratuity amount must be sent

with the initial authorization request. This does note currently apply to the US market, however there is discussion that Mastercard may start disallowing tip modification in the future for Chip cards when using PIN verification (but, presumably, not when using Signature or NoCVM verification).

Please refer to the below card brand documentation for more information:

- http://www.mastercard.com/us/merchant/pdf/TPR-Entire_Manual_public.pdf (page 70)
- https://www.visa.com/chip/merchants/grow-your-business/payment-technologies/credit-card-chip/docs/Play_it_Smart_With_US_Chip_Payment_Transactions.pdf (page 3)

### 5.2.2 What industries are certified for EMV?

EMV doesn't define certifications by industry, the industry-specific data is outside of the EMV-relevant data. Instead EMV works on what they call Terminal Types.

In general, there are 2 relevant terminal types, and those are "Attended" and "Unattended". For "Attended" terminal types, this is where there is a clerk present such as a supermarket, normal retail location, or restaurant. "Unattended" terminal types are used in Self Serve terminals such as kiosks or fuel pumps.

Examples of "Attended" industries in Monetra are "Retail", "Restaurant" and "Lodging".

Examples of "Unattended" industries in Monetra are "Retail Self Serve" and "Automated Fueling".

Please see the Certifications section for what devices and processors are certified for "Attended" vs "Unattended" to see what is currently supported. As long as the industry you support falls into the "Attended" or "Unattended" category that has an EMV certification for the given device and processor you choose, and Monetra itself supports the desired industry for the given processor, then it is a supported configuration.

# 6 UniTerm Protocol Examples



Several examples are provided below which describe how to use the UniTerm protocol.

## 6.1 EMV Transaction [device load]

### 6.1.1 UniTerm Request Data

| PARAMETER | VALUE |
|---|---|
| password | test123 |
| u_action | deviceload |
| u_device | USB |
| u_deviceidlemessage | WELCOME |
| u_devicetype | ingenico_cpx |
| u_flags | DEVICEONLY |
| u_id | 1182112391 |
| username | moneris_ipp320x:sub |

### 6.1.2 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| addltermcaps | F000F0F001 |
| addltermcaps_desired | 6000F0F001 |
| addltermcaps_loa | F000F0A001 |
| altered_termload | yes |
| code | AUTH |
| loa_id | 3C |
| termcaps | E0B8C8 |
| termcaps_desired | E0B8C8 |

| | |
|---|---|
| termcaps_loa | E0B8C8 |
| termtype | 22 |
| termtype_desired | 21 |
| u_errorcode | SUCCESS |
| verbiage | Device loaded |

## 6.2 EMV Transaction [Interac]

### 6.2.1 UniTerm Request Data

| PARAMETER | VALUE |
|---|---|
| action | sale |
| amount | 1.00 |
| nsf | yes |
| ordernum | 899065992 |
| password | test123 |
| u_action | txnrequest |
| u_device | USB |
| u_deviceidlemessage | WELCOME |
| u_devicetype | ingenico_cpx |
| u_flags | DEVICEONLY |
| u_id | 899065992 |
| username | moneris_ipp320x:sub |

### 6.2.2 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXXXXX2145 |
| auth | 221093 |
| batch | 1 |
| cardholdername | Test Card 14 |
| cardtype | INTERAC |
| code | AUTH |
| item | 793 |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |

| | |
|---|---|
| merch_id | 1625 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| msoft_code | INT_SUCCESS |
| pclevel | 0 |
| phard_code | SUCCESS |
| rcpt_acct_type | checking |
| rcpt_custom | refnum:660136000010017930 |
| rcpt_emv_ac | 882D8427A268E214 |
| rcpt_emv_actype | TC |
| rcpt_emv_aid | A0000002771010 |
| rcpt_emv_cvm | pin |
| rcpt_emv_name | Interac |
| rcpt_emv_tsi | 7800 |
| rcpt_emv_tvr | 8000008000 |
| rcpt_entry_mode | C |
| rcpt_host_ts | 072315151022 |
| rcpt_issuer_resp_code | 00 |
| rcpt_resp_code | 001 |
| timestamp | 1437678622 |
| ttid | 992 |
| u_errorcode | SUCCESS |

## 6.3 Pin Debit (forced) Transaction Request

### 6.3.1 UniTerm Request Data

| PARAMETER | VALUE |
|---|---|
| action | sale |
| amount | 1.00 |
| nsf | yes |
| ordernum | 899065992 |
| password | test123 |
| u_action | txnrequest |
| u_cardclass | DEBIT |
| u_device | USB |
| u_deviceidlemessage | WELCOME |

| | |
|---|---|
| u_devicetype | ingenico_rba |
| u_id | 899065992 |
| username | transarmor_isc250:sub |

### 6.3.2 GUI output


### 6.3.3 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXX0027 |
| auth | 412303 |
| batch | 15 |
| cardtype | MCDEBIT |
| code | AUTH |
| item | 139 |
| merch_id | 0993 |
| msoft_code | INT_SUCCESS |
| pclevel | 0 |
| phard_code | SUCCESS |
| rcpt_entry_mode | S |
| timestamp | 1437678765 |
| ttid | 200 |
| u_errorcode | SUCCESS |

# 7 UniTerm Test Application

Included with the UniTerm software distribution is a test application known as "UniTerm Tester". This test application is a simple graphical user interface which may be used to test the various functionality in UniTerm. This utility should be used by developers exploring the functionality of UniTerm as it will provide the request and response messages from UniTerm as well as generate sample receipts for each request. The test utility can be found in the same directory as the `uniterm` executable named `unitermtester`.

# 8 UniTerm Code Examples

Code examples are provided help you understand how easy it is to integrate your application with the UniTerm middleware. Please see Appendix C for complete code examples.

Examples are provided for the following languages:

- Microsoft C# using libmonetra
- Microsoft C# using XML and `HttpWebRequest`
- Java using libmonetra
- PHP using libmonetra
- Microsoft VB.Net using libmonetra
- Microsoft `VBScript` using XML and `MSXML2`
- Microsoft Visual Basic 6 using libmonetra

# 9 UniTerm Hardware Devices (Point of Interaction Devices)



Card data is captured at the point of sale via a magnetic swipe reader or, in some cases (such as for telephone-based transactions), by manual entry of the card number via a keyboard, touch screen, or key pad. The device where card data is captured is called the Point of Interaction (POI) device or also may be referred to as the "point of capture" or "point of entry" device.

Note: The UniTerm module supports both encrypting and non encrypting POI devices. Using the UniTerm module with non encrypting devices can remove the application software (such as a POS application) from scope for the PCI Payment Application (PA-DSS) standard. Using encrypting POI devices can also reduce or eliminate PCI requirements for merchants.

## 9.1 Supported POI Devices

The table below describes POI devices currently supported. The column marked ENCRYPTION indicates the type of encryption the device supports (if any). CardShield encryption can be performed by a Monetra server while other types of encryption must be preformed by the transaction processor. The column marked `EMV` are devices that UniTerm can work with to perform EMV/Chip based transactions.

Note: UniTerm is currently only supporting devices which support EMV. This list may be expanded in the future to support non-EMV devices. This list also does not include keyboard-emulation devices (both encrypting and non-encrypting) which are supported when running in GUI mode.

If you are using a previous version of UniTerm which supported additional non-EMV devices, do not upgrade your version of UniTerm as those devices are not currently supported.

| Model | Device S/W | u_devicetype | Notes | Encryption | EMV |
|---|---|---|---|---|---|
| **Ingenico** | | | | | |
| iPP320 CPX | CPX | ingenico_cpx | Canada, UniTerm v8.0 only | NONE | x |
| iUP250 uCPX | uCPX | ingenico_ucpx | Canada, UniTerm v8.0 only | NONE | x |

| Model | Device S/W | u_devicetype | Notes | Encryption | EMV |
|---|---|---|---|---|---|
| Ingenico RBA family (iPP320, iSC Touch 250, etc) | RBA | ingenico_rba | USA | CardShield, First Data TransArmor RSA | x |
| **Verifone** | | | | | |
| vx805 | XPI | verifone_vx | USA | NONE | x |

### 9.1.1 Ingenico RBA information

The minimum version of the RBA software load supported is v14.0.6 for running EMV transactions. The current recommended version is 19.0.4 if available for your device model, otherwise the latest RBA 18 series for your device should be used (18.0.4 through 18.0.12). Users of UniTerm versions prior to 8.2.0 should continue to use RBA 15.06 or 16.02 (16.02 has a known Mastercard Contactless issue). Please note 15.1X is known to be incompatible, and the entire 15 series should not be used on Bluetooth devices. Any device in the Ingenico RBA family may be used.

Note: RBA v20+ has a new EMV kernel that should not be used at this time as it requires full brand recertification to be used. Future Monetra and UniTerm versions will support the new releases.

The RBA family includes all Ingenico Telium2 devices that can run the RBA (Retail Base Application) software. This includes, but is not limited to:

- iCMP
- iSMP companion
- iWL250
- iPP320 - PCI PTS v3.x+ version
- iPP350 - PCI PTS v3.x+ version
- iSC Touch 250
- iSC Touch 350
- iSC Touch 480

#### 9.1.1.1 Communication Methods

UniTerm supports communicating with RBA via these communication methods (given the proper cables and add-on options from Ingenico):

- USB-HID - No drivers are necessary, select HID as the device connectivity in UniTerm
- USB<>Serial Conv - Requires Telium or Jungo drivers on Windows, will show up as a virtual COM port and be used as a Serial device in UniTerm (Linux and MacOSX do not need drivers, they will show up as serial ports).
- Serial - Settings: 115200 8N1 - No flow control
- Bluetooth - Android native, on Windows it should show up as a virtual COM port and be usable as a Serial device in UniTerm.
- Ethernet - Settings: IP server mode

Note: Not all devices support BlueTooth or `USB-HID` even if the menu lists it as an available option.

### 9.1.1.2 Device configuration

`RBA` devices can be configured by entering the management menu during device boot in order to set up the communication method. When a device is shipped to you, it can often be left in a state which is not compatible with the cabling being used and must be reconfigured. Please ensure you only select one of the supported communication methods as documented in the previous section.

In order to reboot a device, you may either disconnect it from power, or use the reboot key sequence. The key sequence is either the yellow `CLEAR` button plus the "`*.,#`" key or the "`-`" key, depending on which device is being used.

While booting, wait until the `RBA` splash screen appears with the scroll bars and system information. Then quickly press the management password, which by default is `2 6 3 4` and then the green `ENTER` key. Follow the on-screen prompts.

The communication method configuration is available via `TDA -> Configuration -> Communication`.

#### 9.1.1.2.1 Contactless Support

Some devices such as the `iPP320` have contactless as an optional module. It is possible when such a device is shipped to you, the optional module is installed but not activated. If you are certain the device has the necessary hardware for contactless, but contactless is not working, you may need to activate it. For the `iPP320` this can be confirmed by observing the existence of a contactless chip behind the rear door of the device. Contactless must not be enabled if the device does not have the proper hardware.

In order to enable contactless, use the key sequence documented in the prior section to enter management mode. Then navigate to `Telium manager -> Initialization -> Parameters -> Contactless` and make sure it is set to `Yes -> Internal`.

### 9.1.1.3 Hardware Information

It is important to ensure the device being ordered is the latest hardware revision. Ingenico often introduces newer revisions without changing the model number, however their Part Numbers do in fact differ. The easiest way to request the most recent revision is to ensure you are requesting the `PCI PTS v3` or `v4` version of the devices. Older hardware revisions comply with `PCI PTS v2` and should not be used for new deployments as you may experience issues due to limitations in the hardware.

Note: There have been recent reports of customers receiving `iPP320` units that have been sent out as `PCI PTS v2` devices. These devices do NOT support `RBA12` and higher, even though they may come with a later `RBA` release. The part number on the supported devices will start with `11`, while the part number on unsupported devices will start with `01`. If you experience lockups or unexpected behavior, please verify your device is a `PCI PTS v3` or higher device.

#### 9.1.1.4 Forms and Images

UniTerm depends on the stock forms and images that ship by default on terminals with `RBA`. In addition, UniTerm does require a few UniTerm-specific forms and images to be available on the device. These will be generated and uploaded automatically to the device if UniTerm can not find them.

UniTerm will check if it has all the necessary forms on the first transaction run by a device. It will then load any missing forms. When loading forms is required, a message is presented on the device and there is an additional delay until the upload is complete.

It is possible for integrators to fully customize the look and feel of the forms displayed on the device. Such integrators should contact Ingenico and sign up for the developer portal available at https://developer.ingenico.us/ in order to obtain the necessary form building tools. UniTerm provides the `u_action=deviceupload` function to assist integrators in uploading any custom forms and images they have created directly from their POS. Device distributors can also assist with pre-loading forms and images prior to shipping to end users if more convenient. When uploading forms and images, the `u_filename` should only contain the filename and does not need to reference the `HOST` path on the device.

The forms used and their requirements are listed below.

Forms and Images used by UniTerm:

- `UTAD.K3Z` - The form displayed when the device is idle, also known as the "ADs" screen. This form may be customized to present an image or a series of rotating images, but must not contain buttons. The default form loaded contains a single image, `UTAD.PNG` (or `UTAD.BMP` on `iUP250/iUC285` devices). It is recommended that the images created be specific to the device for best appearance even though the device will scale the image if too small or large.

- `UTCCOD.K3Z` - Form used for card entry / selection. The form loaded is the same as the default Ingenico `CCOD.K3Z` form, with the exception that the `cancelenabled='true'` attribute has been added to allow the cardholder to press the physical cancel button to exit the request payment screen. Integrators wishing to modify this screen need to comply with the capabilities of the stock form.

- `UTCSEL.K3Z` - Form used for tender selection (credit, debit, etc). The form loaded is identical to the default Ingenico `PAY1.K3Z` form. It is duplicated due to an Ingenico limitation that does not allow the use of the stock form when using the "on demand" command mode. Integrators wishing to modify this form must comply with the capabilities of the stock form, especially the mapping of the button names available (e.g. `Bbtna` - debit, `Bbtnb` - credit, etc).

- `MSG.K3Z` - Form used to display single line messages. This is a stock form, any replacements should adhere to the capabilities of the stock form.

- `MSGTHICK.K3Z` - Form used to display double line messages. This is a stock form, any replacements should adhere to the capabilities of the stock form.

- `AMTV.K3Z` - Form used to display confirmation prompts, both for arbitrary prompts and amount confirmation. This is a stock form, any replacements should adhere to the capabilities of the stock form.
- `UTASEL.K3Z` - Form used to display tip and Cash Back prompts, based on the `cashba.K3Z` stock form. Button IDs must be:
  - `O` - Other
  - `N` - No (won't be shown on smaller devices)
  - `A` - Amount 1
  - `B` - Amount 2
  - `C` - Amount 3
- Ingenico may internally call additional forms during the EMV payment processing flow. For information on how to customize these screens, integrators should contact Ingenico.

#### 9.1.1.5 First Data `TransArmor` RSA Encryption

The Ingenico devices support First Data's `TransArmor` RSA encryption. `TransArmor` is First Data's P2PE encryption solution along with tokenization which must be enabled on the account both within First Data's systems as well as within Monetra. When configuring a Monetra account for `TransArmor` encryption, set the `Encryption` merchant configuration value to `IngenicoRSA`.

As part of the device loading procedure, a key request will be made to Monetra which will request the current key to use from First Data's systems. Monetra will then send that key identifier to `takeys.monetra.com:443` to look for an available signed package to load onto the Ingenico device. Due to limitations in the Ingenico `TransArmor` implementation it is not possible to directly load the key from First Data's systems into the device. If the requested key package is not yet available, the existing key will be continued to be used until which time the updated package is made available.

`TransArmor` keys typically expire after 2 years, and new keys will be provided 90 days prior to expiration. All terminals on a given merchant account will share the same RSA public key.

#### 9.1.1.6 Updating `RBA` firmware with UniTerm

As of UniTerm v8.2, firmware updating is supported through UniTerm. As part of the `RBA` integration kit provided by Ingenico, there are firmware files provided for multiple upgrade options. The format supported by UniTerm is the `OGZ` format, which is a single-file firmware update. Each device in the `RBA` family has its own firmware file. For instance, an `iSC 250` cannot use the same firmware file as an `iPP320`.

Upgrading should take place via `USB<>Serial Conv` mode if possible. `USB-HID` mode will add significant time (20+ minutes) to the upgrade process. `Ethernet` mode is quick, but it is known that if a device is configured with a static IP address, the firmware update may reset the device to `DHCP`.

The firmware files are not currently provided by Main Street Softworks, and must be obtained directly from Ingenico. This may change in the future to further aid integrators with deployment updates.

Note: Upgrading the firmware can take several minutes and may wipe all custom settings (including encryption), forms, and images. Only firmware upgrading is tested and supported, downgrading is not recommended. It is also only recommended to update to firmware versions which have been approved for use by UniTerm to ensure compatibility. Please ensure power is not unplugged during an upgrade or the device could be required to be sent in for repair.

Provided below is the recommended steps to perform in order to upgrade the `RBA` firmware via UniTerm.

1. Request the current device model and `RBA` version via the `u_action=deviceinfo` command. Ensure the current firmware version needs to be updated before continuing, take note of any additional information returned such as the encryption type.

2. Locate the proper `OGZ` for the desired `RBA` version to load for the device model and send it to UniTerm via the `u_action=deviceupload` command. The upgrade process may take 5 minutes or more depending on the connectivity method used. The device will reboot and apply the update and UniTerm should return once the update is complete.

3. Perform a `u_action=deviceinfo` request to ensure the firmware was successfully updated to the desired version.

4. Perform a `u_action=deviceload` with the `u_forceload=yes` parameter to ensure UniTerm re-downloads all EMV settings to the device. The device will reboot after this process.

5. Upload any custom forms or images to the device using the `u_action=deviceupload` command. These forms or images may be uploaded one at a time, or in bulk by packaging them into a `TGZ` file.

6. Upload any security files such as encryption activation or BIN exclusion lists using the `u_action=deviceupload` command.

   If keys are loaded for Monetra/CardShield encryption, `RBA` includes a file known as `MONETRA.PGZ` or `MONXXYY.PGZ` where the `XXYY` corresponds to the `RBA` version. This file must be re-loaded in order to ensure the device outputs encrypted data otherwise all data will be output unencrypted. The encryption type loaded prior to the upgrade is returned via the `deviceinfo` request in the first step. It is important to ensure that the encryption file used is the one for the specific `RBA` version in use or the device may refuse to boot or otherwise behave in abnormal ways.

7. If any files were uploaded to the device after the `deviceload`, it is necessary to call `u_action=devicereboot` before the device is ready to be used.

### 9.1.2 Verifone VX XPI information

The version of the `XPI` software load supported is v8.23a for running EMV transactions. Prior versions of `XPI` may work for non-EMV transactions, however this functionality has not been extensively tested. Newer versions are known to have issues which prevent them from properly functioning in some EMV test cases.

#### 9.1.2.1 Communication Methods

UniTerm supports communicating with the Verifone VX via these communication methods (given the proper cables and add-on options from Verifone):

- USB - Requires Vx USB Drivers available from www.verifone.com on Windows, will show up as a virtual COM port and be used as a Serial device in UniTerm (MacOSX and Linux do not require a driver, will show up as a serial port).
- Serial (COM1) - 9600

When a device is shipped to you, it can often be left in a state which is not compatible with the cabling being used and must be reconfigured.

#### 9.1.2.2 Device configuration

During device boot-up, it is possible to change the connectivity setting to match the cabling. When the XPI version is displayed during startup, press the alpha and 8 buttons simultaneously. You can then change the connectivity method by pressing the appropriate F<n> key.

### 9.1.3 Ingenico CPX/uCPX information

Ingenico CPX (attended) and uCPX (unattended) software loads are supported for Canadian merchants. These loads support both contact and contactless EMV processing for multiple card brands including Interac debit cards. The required software versions are 10.14 for CPX and 02.02 for uCPX.

Note: The MasterCard PayPass v2.1 kernel must be loaded into the device if supporting contactless MasterCard EMV. If the device is loaded with the PayPass v3.0 kernel, it will fail to accept PayPass transactions. Due to limitations in the Ingenico software, it is impossible for UniTerm to detect the version of the PayPass kernel in use, and the CPX and uCPX software versions are not tied to any PayPass kernel version.

#### 9.1.3.1 Communication Methods

UniTerm supports communicating with CPX/uCPX via these communication methods (given the proper cables and add-on options from Ingenico):

- USB->Serial - Requires Telium or Jungo drivers on Windows, will show up as a virtual COM port.
- Serial - 9600 7bits Even Parity - No flow control
- IP/Ethernet - Even Parity

Note: Please contact Ingenico for assistance with configuring your device for proper communication. It is known that the on-screen menu system does NOT work when configuring Ethernet mode due to the inability to set the Parity to Even. The parity configuration is a crucial step in ensuring Ethernet connectivity is functional.

## 9.2 Obtaining Devices

Obtaining the right device for use with UniTerm requires care to ensure a few major factors:

- Is the device supported by UniTerm?
  - Please see the prior section: Section 9.1
- Is the device certified for the Processing Institution being used?
  - Please see the next chapter: Section 10.1
- Does the device come preloaded with both the right software and version?
  - Please see section specific to the chosen device under Section 9.1 for supported software revisions running on the device.
- Is the vendor the device is sourced from able to load the appropriate keys (Pin Debit or Encryption) as required?
  - Please see the next section: Section 9.2.1

### 9.2.1 Where to source devices with appropriate loads and keys

Most providers of POS hardware also offer loading and injection services. It is recommended that you work with one or more providers that can offer these services to ensure smooth deployments and reduced shipping costs. When a device is ordered, other than the device model being ordered, you must also indicate the software load and version to be installed on the device. Most device manufacturers have multiple software loads available so it is essential that you adhere to any documented software requirements to ensure compatibility.

In addition to the software loads on the device, most devices will require a Pin Debit key be loaded into the device. In the US market this is a `3DES DUKPT` key used for encrypting pins. Though commonly referred to as a Pin Debit key, it is also required for EMV Credit Card Online PIN verification which uses the same infrastructure for verification of PINs. If EMV is being used, and the certification states Online PIN is used, you MUST have a PIN Debit key injected into your device. Unfortunately there are dozens of potential keys, and the correct one for the processing institution and acquirer being used must be injected into the device. You should contact your merchant account provider or acquiring bank to get the Key Serial ID needed for injecting into your device and work with your hardware provider to ensure they have that key available. Only secure key injection facilities can load PIN keys into a device, it is not possible for a merchant to load their own PIN keys.

Note: A single processing institution may process transactions for more than one acquiring bank, and each acquiring bank may mandate their own PIN keys. It is absolutely necessary to not simply rely on obtaining a PIN key that works with your processing institution, but also one that works for the specific acquiring bank. Only your merchant account provider can assist you with identifying who your acquiring bank is and what PIN key identifier should be used.

If Point to Point Encryption is also desired, it is necessary to have your key injection facility also load that key into your device and enable the appropriate encryption support in the software to utilize the key before shipping the device. One notable exception is First Data's RSA `TransArmor` on `Ingenico RBA` devices does not need to be loaded as UniTerm can load the appropriate keys itself. If using CardShield encryption, it is necessary to have first

shared a Base Derivation Key with your key injection facility from your instance of Monetra, in order for them to be able to load the key into your device and enable the support before shipping.

Some device distributors that provide loading and key injection facilities are listed below in alphabetical order:

- JRs POS Depot
- The Phoenix Group
- POS Portal
- Scan Source
- TASQ

# 10 Certifications and Device Configurations



## 10.1 Certification List

EMV Certifications are tied to specific device versions, device configurations, and software versions. During deployment, it is crucial that only certified configurations are used.

Device configurations are based on the EMV kernel version in the device. The available configurations are listed as part of the EMV LOA (Letter of Approval) for the Level 2 kernel for the device. The approval letters can be obtained from EMVCo: http://www.emvco.com/approvals.aspx?id=85

| Device | EMVKern/Conf | UniTerm | Module Version | Config |
|---|---|---|---|---|
| **Chase Paymentech** | | | | |
| Ingenico iPP320 CPX | 4.66/3C | 8.0 only (not supported on 8.2) | Paymentech Tampa 3.2.0 (Done Jan 2015) | Canada, Attended, OfflinePin, Sig |
| Verifone vx805 | 6.2.0/1C | 8.0+ | Paymentech Tampa 3.2.0 (Done Nov 2015) | USA, Attended, OnlinePin, OfflinePin, Sig |
| Ingenico RBA family | 4.67/1C | 8.0+ | Paymentech Tampa 3.2.0 (Done Feb 2016) | USA, Attended, OnlinePin, OfflinePin, Sig |
| **Moneris** | | | | |
| Ingenico iPP320 CPX | 4.66/3C | 8.0 only (not supported on 8.2) | Moneris SPDH 2.0.0 (Done Oct 2015) | Canada, Attended, OfflinePin, Sig |
| Ingenico iUN uCPX | 4.66/15C | 8.0 only (not supported on 8.2) | Moneris SPDH 2.0.0 (Done Oct 2015) | Canada, Unattended, OfflinePin, NoSig |
| **First Data** | | | | |
| Ingenico RBA family | 4.67/1C | 8.3+ | First Data Cardnet or Nashville EDC 4.1.1 (Done Feb 2017) | USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless |

| Device | EMVKern/Conf | UniTerm | Module Version | Config |
|---|---|---|---|---|
| Ingenico RBA family | 4.67/1C | 8.0+ | First Data Cardnet or Nashville EDC 4.1.0 (Done Jan 2016) | USA, Attended, OnlinePin, OfflinePin, Sig |
| Verifone vx805 | 6.2.0/1C | 8.0+ | First Data Cardnet or Nashville EDC 4.1.0 (Done Feb 2016) | USA, Attended, OnlinePin, OfflinePin, Sig |
| **Tsys** | | | | |
| Ingenico RBA family | 4.67/1C | 8.0+ | TSYS (aka Vital/VisaNet) 3.0.0 (Done Feb 2016) | USA, Attended, OnlinePin, OfflinePin, Sig |
| **Global Payments** | | | | |
| Ingenico RBA family | 4.67/1C | 8.0+ | Global Payments East 3.0.0 (Done Nov 2015) | USA, Attended, OnlinePin, OfflinePin, Sig |
| Verifone vx805 | 6.2.0/1C | 8.0+ | Global Payments East 3.0.0 (Done Nov 2015) | USA, Attended, OnlinePin, OfflinePin, Sig |
| **WorldPay** | | | | |
| Ingenico RBA family | 4.67/1C | 8.0+ | RBS WorldPay TCMP 2.0.0 (Done Feb 2016) | USA, Attended, OnlinePin, OfflinePin, Sig |
| **Vantiv** | | | | |
| Ingenico RBA family | 4.67/1C | 8.0+ | Vantiv/ FifthThird 610 2.1.0 (Done Jan 2016) | USA, Attended, OnlinePin, OfflinePin, Sig |

## 10.2 Configuration Definitions

EMV configurations are strictly certified in an "all or nothing" manner. You must choose an explicit certification from the list in the prior section and all configuration parameters must be adhered to. For instance, if the certification lists both `OnlinePin` and `OfflinePin`, you cannot simply choose to support only `OfflinePin`.

The meanings of the various configurations listed in the prior section are below:

| Key | Description |
|---|---|
| `USA` | Certified for use in the United States |
| `Canada` | Certified for use in Canada |
| `Attended` | The environment is monitored by a clerk such as Retail, Restaurant, or Lodging. Not usable in a Kiosk environment such as a parking meter or gas pump. |
| `Unattended` | The environment is NOT monitored by a clerk, for use in kiosk type environments. |
| `OnlinePin` | An encrypted PIN can be obtained from a cardholder and sent to the processor with the transaction. When supporting Online PIN it is required that the device be injected with a `3DES DUKPT` PIN key specific to the processing institution in use prior to deployment by a merchant.<br>  Note: `OnlinePin` may not be supported for all card brands of a given processing institution. UniTerm will automatically adjust support for the processor's card brand limitations where necessary. |
| `OfflinePin` | The terminal will negotiate the PIN directly with the chip embedded into the card without the need to send the PIN to the processing institution. A terminal does not need a Pin Debit key injected into it if only `OfflinePin` (and not `OnlinePin`) is supported. |
| `Sig` | Signature cardholder verification is supported. This may either be a signature capture capable device or a signature obtained via paper receipt. |

# A UniTerm Device Loading

When loading a device with UniTerm, UniTerm will send Monetra a list of terminal configurations from the Letter of Approval (LOA) as provided by device manufacturer for the device's EMV kernel. Monetra will compare this list to merchant defined settings. Monetra will then select a usable LOA configuration and return to UniTerm loading data which has been merged with the merchant's settings.

Some terminal loading data is mandatory and cannot deviate from a LOA configuration. Other data is merchant configurable and is allowed to be changed. Data that is configurable will be merged into an LOA configuration by Monetra based on the merchant's settings.

In the event no LOA configuration is valid for the merchant's settings then Monetra will respond with an error. Also, If the device's EMV kernel version is not certified for use with UniTerm loading will result in an error.

After a successful load the integration must check altered_termload. If it is "yes" then not all of the merchant's settings could be used and some of the values have been ignored. The integration can compare the selected values with the *_desired and *_loa values to determine what was ignored. It is the choice of the integration to either accept the load with the selected values or return an error if the merchant's setting have been altered due to being unsupported by the devices LOA configurations.

Note: If using implicit/auto device loading and not calling `u_action=deviceload` directly, an integrator will have no ability to retrieve the deviceload parameters.

| PARAMETER | OVERVIEW |
|---|---|
| `altered_termload` | If no LOA configuration matches the merchant's settings a valid LOA will be used and the merchant's settings will be overridden. This indicates this has happened. |
| `termtype_desired` | The terminal type Monetra has determined fits the merchant's settings. |
| `termcaps_desired` | Terminal capabilities configured in Monetra. These are features that the merchant has selected for use. |
| `addltermcaps_desired` | Additional terminal capabilities configured in Monetra. These are features that the merchant has selected for use. |
| `termcaps_loa` | Terminal capabilities from the LOA configuration Monetra has selected. |
| `addltermcaps_loa` | Additional terminal capabilities from the LOA configuration Monetra has selected. |
| `loa_id` | The LOA configuration id Monetra has selected for use. This is the id in the device certification document for the kernel version located at: http://www.emvco.com/approvals.aspx?id=85 |
| `termtype` | The terminal type from the LOA configuration that will be loaded into the device. |

| | |
|---|---|
| `termcaps` | Terminal capabilities from the merged LOA configuration and merchant's settings that will be loaded into the device. Note: mandatory LOA configuration data will not be changed. |
| `addltermcaps` | Additional terminal capabilities from the merged LOA configuration and merchant's settings that will be loaded into the device. Note: mandatory `LOA` configuration data will not be changed. |

Example device load response:

```
u_errorcode = SUCCESS
code = AUTH
verbiage = Device loaded
altered_termload = no
termtype_desired = 21
termcaps_desired = E0B8C8
addltermcaps_desired = 6000F0F001
termcaps_loa = 60B8C8
addltermcaps_loa = 6000F0A001
loa_id = 18C
termtype = 22
termcaps = E0B8C8
addltermcaps = 6000F0F001
```

# B EMV Receipt Requirements

The UniTerm application never generate receipts. It is the integrator's responsibility to generate all proper receipts for both customer and merchant retention. What constitutes a proper receipt is dependent on a number of factors such as industry, card present vs card not present, and card entry method (for card present).

The purpose of this section is to provide general information about the receipt data UniTerm will return and how to use it generate a receipt. This does not cover all aspects of receipt generation. It also does not cover processor specific formatting requirements. It is recommended to verify receipts and receipt formatting with your processor before going into production.

Also there are typically two types of receipts printed. A merchant and a customer copy. Each one will have most of the same information but there are slight differences between the two.

## B.1 Receipt content

### B.1.1 Base receipt content

Receipts should include the following blocks and data elements in roughly the order provided below. All data is required if returned by UniTerm, or otherwise available, unless otherwise noted.

- Merchant Info Header
  - Name - `merch_name`
  - Address - `merch_addr1, merch_addr2, merch_addr3`
  - Phone (optional) - `merch_phone`
  - Email (optional) - `merch_email`
  - Website (optional) - `merch_url`
  - Merchant ID - required by some processors. Recommended to omit or truncate, see `merch_id` response documentation for more information.
  - Lane ID (optional) - `laneid` or `stationid` request parameter.
- Transaction type - request parameter `action` or equivalent text
- Card information
  - Type - `cardtype`
  - Entry mode - `rcpt_entry_mode` - or equivalent text, some processors may have explicit mappings they require.
  - Interac Account Type - `rcpt_acct_type` or for Interac Flash (contactless) transactions, must display `INTERAC FLASH DEFAULT`. Integrators must convert the UniTerm-returned value of `checking` to `chequing` to comply with Interac requirements.
  - Masked Account Number - `account`
- Transaction reference info
  - Date and time - `rcpt_host_ts` or `timestamp`
  - Identifier - request parameters `ordernum` or `ptrannum`
  - Additional identifiers (optional) - request parameters such as `custref`
  - `ttid` (optional) - either request or response `ttid`
  - Batch number - `batch`

    - Auth number (if authorized) - `auth`
    - Trace information - `stan`
    - Processor response code (some processors may require this) - `rcpt_resp_code`
    - Issuer response code (some processors may require this) - `rcpt_issuer_resp_code`
- Processor specific custom data - see `rcpt_custom`
- Monetary amounts
    - Tip - Request parameter `examount`
    - Tax - Request parameter `tax`
    - Cash back - Request parameter `cashbackamount`
    - Authorized Amount - `authamount` if returned, otherwise request parameter `amount`
    - Balance - `balance`
- Transaction disposition
    - Card disposition - See Card Disposition documentation
    - Partial Approval Indicator - if `authamount` returned
    - Overall disposition (approved/declined) - `code`
- Additional Print Data - `printdata`, Additional data meant to be printed on the receipt as provided by the processor. Often used for gift/loyalty programs.
- Cardholder Verification
    - Signature line (if necessary) - `u_need_signature=yes`
    - Cardholder Name, centered under signature line if a signature line is shown (optional) - `cardholdername`
- EMV data
    - Application name - `rcpt_emv_name`
    - `AID` - `rcpt_emv_aid`
    - `TVR` - `rcpt_emv_tvr`
    - `TSI` - `rcpt_emv_tsi`
    - Application Cryptogram Type and Cryptogram Value (optional) - `rcpt_emv_actype` and `rcpt_emv_ac`
- Cardholder Notice (such as stating merchant vs customer copy) (optional) - see receipt examples

## B.2 Receipt Data Returned by UniTerm

| PARAMETER | OVERVIEW |
|---|---|
| `timestamp` | Unix timestamp representing the time and date the transaction took place, this should be used to derive the transaction date if `rcpt_host_ts` is not returned. |
| `rcpt_host_ts` | (REQUIRED): The time and date recorded from the processor the transaction took place. `MMDDYYHHMMSS` format. Use `timestamp` if this value is not present in the response. |
| `rcpt_entry_mode` | (REQUIRED): Indicates how the card data was captured. Possible values are:<br><br>• `G:` Keyed entry (EMV Fallback)<br>• `M:` Keyed entry<br>• `T:` EMV Contactless |

| | |
|---|---|
| | • C: EMV Contact<br>• F: Swipe (EMV Fallback)<br>• R: MSD (RFID) Contactless<br>• S: Swipe<br>• I: MICR Check Read |
| rcpt_acct_type | Interac specific account type chosen by the customer. |
| rcpt_emv_cvm | For EMV transactions this is the cardholder verification method performed. Possible values are:<br><br>• none<br>• sig<br>• pin<br>• pinsig<br>• skipped - Used for some contactless transactions below the CVM floor.<br>• cdcvm - Card Holder Device verification. The user's device (usually smart phone) validated the user's identity.<br>• unknown<br><br>For "pin" and "pinsig" the receipt should say "VERIFIED BY PIN". For "sig" a signature should be captured. |
| rcpt_resp_code | Response code returned from the processing institution. |
| rcpt_issuer_resp_code | Response code returned from the issuer. |
| language | Cardholder's language preference. The receipt should be created using this language if possible and shall contain the 2 character ISO language code. |
| batch | The batch number associated with the transaction. |
| cardtype | Monetra cardtype value. This is the value that would have been configured in supported card types for the account. Use this to take card specific action in receipt generation. |
| balance | Current balance on the card after the transaction. |
| rcpt_emv_aid | Card Application ID (AID) used |
| rcpt_emv_name | Textual name of card application used. |
| rcpt_emv_tvr | Transaction verification results. |
| rcpt_emv_tsi | Transaction status information. |

| | |
|---|---|
| rcpt_emv_actype | (optional). Application Cryptogram type.<br>• AAC - Application Authentication Cryptogram (decline)<br>• ARQC - Application Request Cryptogram (intermediate or contactless)<br>• TC - Transaction Certificate (offline or final approval) |
| rcpt_emv_ac | (optional). Application Cryptogram. |
| code | Used to determine if the transaction was approved or declined. |
| account | Masked account number. |
| cardholdername | Customers name as encoded on the card. |
| auth | Authorization code. |
| stan | Processor system trace information (mainly used for pin-debit transactions). |
| authamount | If the amount authorized is different than the requested amount this is the amount that must show on the receipt. It is possible that the integration could pool multiple transactions on one receipt and in that case the authamount needs to be present for each card along with other card specific receipt data. Note that some processors do not allow pooling card data onto one receipt and require separate receipts per card. |
| rcpt_custom | List of comma separated key:value pairs with additional processor specific data that needs to appear on the receipt. |
| u_errorcode | On failure this will provide some information about the failure. Specifically important to receipt processing are the EMV_CARD_REMOVED and EMV_CARD_DENY values. |
| u_need_signature | Used to determine if a signature line is required. |
| printdata | Additional processor-provided data returned by some processors that is intended to be printed on receipts. Often used for Gift/Loyalty programs. Please consult with your processor for more information. |
| issuer_decline | Boolean (yes/no). Currently this value is only returned by Moneris, and is used to indicate if a decline was due to an issuer decline or a local processor decline. The purpose of this response |

| | |
|---|---|
| | parameter is that `Moneris` has different receipt messaging requirements based on who declined the transaction. |
| `merch_name` | Merchant Name if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_addr1` | Merchant Address Line 1 if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_addr2` | Merchant Address Line 2 if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_addr3` | Merchant Address Line 3 if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_phone` | Merchant Phone Number if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_email` | Merchant Contact Email if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_url` | Merchant URL or Website if configured in merchant profile. Cached by UniTerm from `merchinfo` request and sent on every transaction response. |
| `merch_id` | Merchant ID truncated to only the last 4 digits if available. Cached by UniTerm from `merchinfo` request and sent on every transaction response. The Merchant ID is required by some processors for EMV, though due to rampant "return fraud", we strongly discourage integrators from providing the full Merchant ID on receipts. Instead, if you choose to display the merchant id, it should display only the last 4 digits. This field can be used for that purpose. |
| `merch_proc` | Merchant Processing Institution (internal name) used. Cached by UniTerm from `merchinfo` request and sent on every transaction response. This may be used to trigger different |

| | |
|---|---|
| | receipt formats based on processor-specific requirements. |

## B.3 Receipt Data NOT Returned by UniTerm

This is information is data that may have been sent to UniTerm on the request that should be on the receipt.

| PARAMETER | OVERVIEW |
|---|---|
| `Transaction Type` | The initiating application should know which transaction type is being preformed (Sale, Refund etc.). |
| `Transaction Identifier` | `ordernum or ptrannum` if present. |
| `Additional Identifier` | `custref` if present. |
| `ttid` | When performing a transaction such as return by `ttid` the referenced `ttid` should be present on the receipt. This will aid in tracking the original transaction that was returned. |
| `Amount Information` | • `Tip` - Tip amount for order as provided in the `examount` field in request.<br>• `Tax` - Tax amount for order<br>• `Amount` - Authorized amount, either the amount passed in or the partially approved amount provided in the `authamount` field.<br>• `Cash Back Amount` - Amount of Cash Back |
| `Card Disposition` | • When `u_errorcode` is `EMV_CARD_REMOVED`, should say: "CARD REMOVED"<br>• When `u_errorcode` is `EMV_CARD_DENY`, should say: "DECLINED BY CARD"<br>• When `rcpt_emv_cvm` is `pin` or `pinsig`, should say: "VERIFIED BY PIN"<br>• When `rcpt_entry_mode` is `F` or `G`, should say: "CHIP CARD SWIPED"<br>• When `authamount` is returned and is not equal to requested amount, should say: "TRANSACTION PARTIALLY APPROVED"<br>• When `code` is `CALL`, should say: "CALL ISSUER" |

## B.4 Signature Line Requirements

The only time a signature line is necessary when using UniTerm is when u_need_signature=yes. Internally UniTerm will handle logic to determine if the signature line is needed on the paper receipt.

When set to yes this indicates that a signature line is required on the receipt. If possible UniTerm will attempt to capture the signature thought the device. If this fails or is not possible then this value will indicate that signature is still required.

## B.5 Merchant vs Customer Copy

For the most part merchant and customer receipt requirements are identical, though there are a couple of minor exceptions.

Merchant receipts must NOT contain a balance line

Customer receipt must not contain a signature line

## B.6 `Moneris` Requirements

`Moneris` has additional receipt requirements that are not covered by this section due to direct contradictions with requirements as provided by other processing institutions and the card brands themselves. The receipt requirements documented are insufficient to comply with `Moneris` requirements but do comply with the card brand requirements. The additional requirements imposed are specific to `Moneris` and appear to be arbitrary, a large enough merchant might be able to negotiate different receipt formats since there is no industry regulation being followed.

If intending to work with `Moneris`, it is required that integrators create a custom receipt template specific to `Moneris` that is used only on `Moneris`, and a separate template that is used for all other processors. Integrators must contact `Moneris` directly to receive their receipt formatting requirements. UniTerm does return sufficient data to format the `Moneris`-specific receipts, it simply may require some data to be manipulated, formatted, or translated to different languages to comply with their requirements.

## B.7 Receipt Examples

Main Street successfully certified EMV, across several processors, using the examples provided below. Note these examples were designed to format properly on a common 25 character receipt printer.

Note: Receipt requirements required for the card brands for EMV and various processors tend to be very strict. We strongly recommend integrators make their receipts resemble those of the examples as closely as possible. Any divergence from the receipt examples provided below may require you seek validation of such receipts from your processor.

### B.7.1 EMV Insert, Signature Required

#### B.7.1.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXX0119 |
| auth | 152013 |
| batch | 1 |
| cardholdername | VISA ACQUIRER TEST/CARD 01 |

| | |
|---|---|
| cardlevel | VISA_TRADITIONAL |
| cardtype | VISA |
| code | AUTH |
| item | 27 |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1834 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| merch_proc | GLOBALPAY |
| msoft_code | INT_SUCCESS |
| pclevel | 0 |
| phard_code | SUCCESS |
| rcpt_custom | REC #:000027,TRN REF #:355724280069888,VAL CODE:BBCD |
| rcpt_emv_ac | 8F73ED36C8F2C099 |
| rcpt_emv_actype | TC |
| rcpt_emv_aid | A0000000031010 |
| rcpt_emv_cvm | sig |
| rcpt_emv_name | CREDITO DE VISA |
| rcpt_emv_tsi | F800 |
| rcpt_emv_tvr | 0280008000 |
| rcpt_entry_mode | C |
| rcpt_host_ts | 092215174640 |
| rcpt_issuer_resp_code | 00 |
| rcpt_resp_code | 000 |
| stan | 378222 |
| timestamp | 1442944083 |
| ttid | 29 |
| u_errorcode | SUCCESS |
| u_need_signature | yes |
| verbiage | AP |

#### B.7.1.2 Example Receipt

```
     MERCHANT NAME
    123 STREET NAME
```

```
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1834         Lane: 1
VISA                    C
Card:    XXXXXXXXXXXX0119
Time:   09/22/15 17:46:40
Order #:              1842
TTID:                   29
RespCode:           00/000
Auth: 152013     Batch: 1
STAN:                378222
REC #:               000027
TRN REF #:355724280069888
VAL CODE:             BBCD

AMOUNT:               1.00

        APPROVED

SIGNATURE

X_____________________________
 VISA ACQUIRER TEST/CARD
           01

CARDHOLDER WILL PAY CARD
   ISSUER ABOVE AMOUNT
 PURSUANT TO CARDHOLDER
        AGREEMENT

CREDITO DE VISA
AID A0000000031010
TVR 0280008000
TSI F800
TC 8F73ED36C8F2C099

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1834         Lane: 1
VISA                    C
Card:    XXXXXXXXXXXX0119
Time:   09/22/15 17:46:40
Order #:              1842
TTID:                   29
RespCode:           00/000
Auth: 152013     Batch: 1
STAN:                378222
REC #:               000027
TRN REF #:355724280069888
```

```
VAL CODE:            BBCD

AMOUNT:              1.00

        APPROVED

CREDITO DE VISA
AID A0000000031010
TVR 0280008000
TSI F800
TC 8F73ED36C8F2C099

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

## B.7.2 EMV Insert, PIN Verified

### B.7.2.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXX0036 |
| auth | 602664 |
| batch | 1 |
| cardholdername | VISA ACQUIRER TEST/CARD 03 |
| cardlevel | VISA_TRADITIONAL |
| cardtype | VISA |
| code | AUTH |
| item | 7 |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1834 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| merch_proc | GLOBALPAY |
| msoft_code | INT_SUCCESS |
| pclevel | 0 |
| phard_code | SUCCESS |
| rcpt_custom | REC #:000007,TRN REF #:638114437174992,VAL CODE:BBCD |
| rcpt_emv_ac | 2F0346EBCA494BF4 |
| rcpt_emv_actype | TC |
| rcpt_emv_aid | A0000000031010 |
| rcpt_emv_cvm | pin |
| rcpt_emv_name | CREDITO DE VISA |
| rcpt_emv_tsi | F800 |
| rcpt_emv_tvr | 0080008000 |
| rcpt_entry_mode | C |
| rcpt_host_ts | 092215172258 |
| rcpt_issuer_resp_code | 00 |
| rcpt_resp_code | 000 |
| stan | 563536 |

| | |
|---|---|
| timestamp | 1442942662 |
| ttid | 7 |
| u_errorcode | SUCCESS |
| verbiage | AP |

### B.7.2.2 Example Receipt

```
      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1834         Lane: 1
VISA                   C
Card:    XXXXXXXXXXXX0036
Time:   09/22/15 17:22:58
Order #:             5705
TTID:                   7
RespCode:          00/000
Auth: 602664     Batch: 1
STAN:              563536
REC #:             000007
TRN REF #:638114437174992
VAL CODE:            BBCD

AMOUNT:            337.00

     VERIFIED BY PIN

        APPROVED

CREDITO DE VISA
AID A0000000031010
TVR 0080008000
TSI F800
TC 2F0346EBCA494BF4

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1834         Lane: 1
VISA                   C
Card:    XXXXXXXXXXXX0036
Time:   09/22/15 17:22:58
Order #:             5705
TTID:                   7
RespCode:          00/000
```

```
Auth: 602664     Batch: 1
STAN:              563536
REC #:             000007
TRN REF #:638114437174992
VAL CODE:            BBCD

AMOUNT:            337.00

     VERIFIED BY PIN

        APPROVED

CREDITO DE VISA
AID A0000000031010
TVR 0080008000
TSI F800
TC 2F0346EBCA494BF4

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

### B.7.3 EMV Insert, No CVM

#### B.7.3.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXX1005 |
| auth | 232508 |
| batch | 1 |
| cardholdername | AEIPS 32/VER 1.0 |
| cardtype | AMEX |
| code | AUTH |
| item | 2 |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1836 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| merch_proc | GLOBALPAY |
| msoft_code | INT_SUCCESS |
| pclevel | 0 |
| phard_code | SUCCESS |
| rcpt_custom | REC #:000002,TRN REF #:416237190201752 |
| rcpt_emv_ac | 5C221DC28EB72FCF |
| rcpt_emv_actype | TC |
| rcpt_emv_aid | A000000025010801 |
| rcpt_emv_cvm | none |
| rcpt_emv_name | AMERICAN EXPRESS |
| rcpt_emv_tsi | F800 |
| rcpt_emv_tvr | 0000008000 |
| rcpt_entry_mode | C |
| rcpt_host_ts | 092515194045 |
| rcpt_issuer_resp_code | 000 |
| rcpt_resp_code | 000 |
| stan | 000514 |
| timestamp | 1443210133 |

| | |
|---|---|
| ttid | 79 |
| u_errorcode | SUCCESS |
| verbiage | AP |

### B.7.3.2 Example Receipt

```
      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1836         Lane: 1
AMEX                   C
Card:     XXXXXXXXXXX1005
Time:   09/25/15 19:40:45
Order #:               41
TTID:                  79
RespCode:         000/000
Auth: 232508     Batch: 1
STAN:              000514
REC #:             000002
TRN REF #:416237190201752

AMOUNT:             62.00

        APPROVED

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008000
TSI F800
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1836         Lane: 1
AMEX                   C
Card:     XXXXXXXXXXX1005
Time:   09/25/15 19:40:45
Order #:               41
TTID:                  79
RespCode:         000/000
Auth: 232508     Batch: 1
STAN:              000514
REC #:             000002
TRN REF #:416237190201752
```

```
AMOUNT:             62.00

        APPROVED

AMERICAN EXPRESS
AID A0000000025010801
TVR 0000008000
TSI F800
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

## B.7.4 EMV Insert, Card Decline

### B.7.4.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXX0010 |
| cardtype | VISA |
| code | DENY |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1834 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| merch_proc | GLOBALPAY |
| rcpt_emv_ac | BA9BD3FAC8ADD6C7 |
| rcpt_emv_actype | AAC |
| rcpt_emv_aid | A0000000031010 |
| rcpt_emv_cvm | pin |
| rcpt_emv_name | CREDITO DE VISA |
| rcpt_emv_tsi | E800 |
| rcpt_emv_tvr | 0280A08000 |
| rcpt_entry_mode | C |
| rcpt_host_ts | 092215134154 |
| u_errorcode | EMV_CARD_DENY |
| verbiage | Transaction aborted - declined by card |

### B.7.4.2 Example Receipt

```
      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1834         Lane: 1
VISA                   C
Card:    XXXXXXXXXXXX0010
Time:   09/22/15 13:41:54
Order #:            17421

AMOUNT:             22.00
```

```
    DECLINED BY CARD
     VERIFIED BY PIN

        DECLINED

CREDITO DE VISA
AID A0000000031010
TVR 0280A08000
TSI E800
AAC BA9BD3FAC8ADD6C7

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1834         Lane: 1
VISA                   C
Card:    XXXXXXXXXXXX0010
Time:   09/22/15 13:41:54
Order #:            17421

AMOUNT:             22.00

    DECLINED BY CARD
     VERIFIED BY PIN

        DECLINED

CREDITO DE VISA
AID A0000000031010
TVR 0280A08000
TSI E800
AAC BA9BD3FAC8ADD6C7

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

## B.7.5 EMV Insert, Card Removed (Decline)

### B.7.5.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| code | DENY |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1818 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| merch_proc | GLOBALPAY |
| u_errorcode | EMV_CARD_REMOVED |
| verbiage | Card Removed |

### B.7.5.2 Example Receipt

```
      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1818         Lane: 1
Time:   09/24/15 14:43:46
Order #:             6224

AMOUNT:              1.00

      CARD REMOVED

        DECLINED


 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1818         Lane: 1
Time:   09/24/15 14:43:46
Order #:             6224
```

```
AMOUNT:              1.00

      CARD REMOVED

        DECLINED


 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

### B.7.6 EMV Insert, Interac

#### B.7.6.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXX1933 |
| auth | 175180 |
| avs | UNKNOWN |
| batch | 1 |
| cardholdername | Test Card 1 |
| cardtype | INTERAC |
| code | AUTH |
| item | 10 |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 3636 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| merch_proc | PAYMENTECH |
| msoft_code | INT_SUCCESS |
| pclevel | 0 |
| phard_code | SUCCESS |
| rcpt_acct_type | checking |
| rcpt_emv_ac | 882D8427A268E214 |
| rcpt_emv_actype | TC |
| rcpt_emv_aid | A0000002771010 |
| rcpt_emv_cvm | pin |
| rcpt_emv_name | Interac |
| rcpt_emv_tsi | 7800 |
| rcpt_emv_tvr | 8000008000 |
| rcpt_entry_mode | C |
| rcpt_host_ts | 092515155118 |
| rcpt_resp_code | A |
| stan | 00298722 |
| timestamp | 1443210676 |

| | |
|---|---|
| ttid | 10 |
| u_errorcode | SUCCESS |
| verbiage | APPROVED |

### B.7.6.2 Example Receipt

```
      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 3636         Lane: 1
INTERAC                C
Acct Type:       CHEQUING
Card:    XXXXXXXXXXXX1933
Time:   09/25/15 15:51:18
Order #:        899065992
TTID:                  10
RespCode:               A
Auth: 175180     Batch: 1
STAN:            00298722

AMOUNT:              5.01

     VERIFIED BY PIN

        APPROVED

Interac
AID A0000002771010
TVR 8000008000
TSI 7800
TC 882D8427A268E214

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 3636         Lane: 1
INTERAC                C
Acct Type:       CHEQUING
Card:    XXXXXXXXXXXX1933
Time:   09/25/15 15:51:18
Order #:        899065992
TTID:                  10
RespCode:               A
Auth: 175180     Batch: 1
STAN:            00298722
```

```
AMOUNT:              5.01

     VERIFIED BY PIN

        APPROVED

Interac
AID A0000002771010
TVR 8000008000
TSI 7800
TC 882D8427A268E214

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

## B.7.7 EMV Contactless, Interac Flash Decline

### B.7.7.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXXXXX1311 |
| cardtype | INTERAC |
| code | DENY |
| issuer_decline | yes |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1625 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| msoft_code | INT_SUCCESS |
| phard_code | GENERICFAIL |
| printdata | CARD CANCELLED*REFER TO BRANCH |
| rcpt_acct_type | flash |
| rcpt_custom | refnum:660136000010016710 |
| rcpt_emv_ac | ED538D29D3390729 |
| rcpt_emv_actype | ARQC |
| rcpt_emv_aid | A0000002771010 |
| rcpt_emv_cvm | unknown |
| rcpt_emv_name | Interac |
| rcpt_emv_tvr | 0080008000 |
| rcpt_entry_mode | T |
| rcpt_host_ts | 072015180303 |
| rcpt_issuer_resp_code | 05 |
| rcpt_resp_code | 058 |
| sequenceid | 671 |
| timestamp | 1437429783 |
| ttid | 861 |
| u_errorcode | MONETRA_ERROR |
| verbiage | DECLINED * CARD CANCELLED |

### B.7.7.2 Example Receipt

```
      MERCHANT NAME
     123 STREET NAME
     CITY, STATE ZIP
     (888) 555-1234

          SALE

MID: 1625         Lane: 1
INTERAC                T
Acct Type:  FLASH DEFAULT
Card: XXXXXXXXXXXXXX1311
Date/Time:   072015180303
Order #:        899065992
TTID:                861
refnum:660136000010016710

AMOUNT:             1.09

        DECLINED

Interac
AID A0000002771010
TVR 0080008000
ARQC ED538D29D3390729

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/CUSTOMER COPY
```

## B.7.8 EMV Contactless, Decline

### B.7.8.1 UniTerm Response Data

| PARAMETER | VALUE |
|---|---|
| account | XXXXXXXXXXXX0010 |
| cardholdername | ETEC/PAYPASS |
| cardtype | MC |
| code | DENY |
| language | en |
| merch_addr1 | 123 STREET NAME |
| merch_addr2 | CITY, STATE ZIP |
| merch_id | 1625 |
| merch_name | MERCHANT NAME |
| merch_phone | (888) 555-1234 |
| msoft_code | INT_SUCCESS |
| phard_code | GENERICFAIL |
| rcpt_custom | refnum:660136000010016700 |
| rcpt_emv_ac | 16D1284D85A29DF2 |
| rcpt_emv_actype | ARQC |
| rcpt_emv_aid | A0000000041010 |
| rcpt_emv_cvm | none |
| rcpt_emv_name | PPC MCD 01 v2 2 |
| rcpt_emv_tvr | 0000008000 |
| rcpt_entry_mode | T |
| rcpt_issuer_resp_code | 51 |
| rcpt_resp_code | 481 |
| sequenceid | 670 |
| timestamp | 1437429662 |
| ttid | 860 |
| u_errorcode | MONETRA_ERROR |
| verbiage | DECLINED * |

### B.7.8.2 Example Receipt

```
     MERCHANT NAME
    123 STREET NAME
    CITY, STATE ZIP
    (888) 555-1234
```

```
          SALE

MID: 1625         Lane: 1
MC                     T
Card:    XXXXXXXXXXX0010
Date/Time:   072015180102
Order #:        899065992
TTID:                860
refnum:660136000010016700

AMOUNT:             10.51

        DECLINED

PPC MCD 01 v2 2
AID A0000000041010
TVR 0000008000
ARQC 16D1284D85A29DF2

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/CUSTOMER COPY
```

# C UniTerm Code Examples

## C.1 Microsoft C# using libmonetra

```
/* UniTerm example program in C#
 *
 * Depends on the libmonetra C# .Net native API
 *
 * Implemented based on the UniTerm Guide in conjunction with the
 * Monetra Client Interface Protocol Specification
 *
 * Please contact support@monetra.com with any questions
 */
using System;
using System.Collections;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading;
using libmonetra;

/* NOTE: if compiling with Mono, you can use
 *        gmcs /unsafe utest.cs libmonetra.cs
 */

class UTest {
 /* Uniterm Connectivity Information
  * NOTE: this is the default, it is possible to change, but 99%
  *       of deployments will probably use this Uniterm information
  *       as-is
  */
 private const string uniterm_host     = "localhost";
 private const int    uniterm_port     = 8123;

 /* Authentication information
  * NOTE: This information corresponds with the public test server
  *       at testbox.monetra.com:8665 */
 private const string monetra_user     = "test_retail:public";
 private const string monetra_pass     = "publ1ct3st";


 static string uniterm_path()
 {
  switch (Environment.OSVersion.Platform) {
   case PlatformID.Win32NT:
   case PlatformID.Win32S:
   case PlatformID.Win32Windows:
   case PlatformID.WinCE:
    return "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe";
   default:
    return "/usr/local/uniterm/bin/uniterm";
  }
 }

 /*! Function to launch the Uniterm from the current process.
```

```
52     *  If we don't launch it from the current process, it won't be given
53     *  focus! (at least on Windows this is true, until the first
54     *  manual focus is performed by an end-user) */
55    static void uniterm_launch()
56    {
57     Process uniterm                 = new Process();
58     uniterm.StartInfo.FileName      = uniterm_path();
59     uniterm.StartInfo.CreateNoWindow = true;
60
61     uniterm.Start();
62
63     /* Make sure Uniterm is ready before returning,
64      * Sleep 1000ms (1s) */
65     System.Threading.Thread.Sleep(1000);
66    }
67
68
69    /*! Function to connect to an endpoint which uses the standard 'monetra'
70     * style protocol (so either Monetra itself, or Uniterm)
71     * \param[in]  host     Resolvable hostname or IP address to connect to
72     * \param[in]  port     Port associated with hostname to establish an SSL
73     *                      connection to
74     * \param[out] errorstr Textual error message if returns null
75     * \return Initialized connection class on success. null on failure
76     */
77    static Monetra uniterm_connect_host(string host, int port, ref string errorstr)
78    {
79     /* Initialize the Class */
80     Monetra conn = new Monetra();
81
82     errorstr = "";
83
84     /* We always want to use an SSL connection to Monetra and Uniterm */
85     conn.SetSSL(host, port);
86
87     /* Do not verify the SSL certificate, Monetra and the Uniterm
88      * use self-signed certificates by default which cannot be validated.
89      * The connection is still encrypted, the endpoint just isn't strictly
90      * validated */
91     conn.VerifySSLCert(false);
92
93     /* This makes it so TransSend() will block until a response is
94      * received from Monetra.  Simplifies the API since we will never
95      * have more than one outstanding transaction per connection in
96      * this application */
97     conn.SetBlocking(true);
98
99     /* Connect! */
100    if (!conn.Connect()) {
101     errorstr = conn.ConnectionError();
102     return null;
103    }
104
105    return conn;
106   }
107
108
109   /*! Wrapper function to connect to Uniterm
```

```
110     * \param[out] errorstr Textual error message if returns null
111     * \return Initialized connection class on success. null on failure
112     */
113    static Monetra uniterm_connect(ref string errorstr)
114    {
115     Monetra conn;
116     string myerror = "";
117     conn = uniterm_connect_host(uniterm_host, uniterm_port, ref myerror);
118     if (conn == null) {
119      errorstr = "Connection to Uniterm Failed: " + myerror;
120     }
121     return conn;
122    }
123
124
125    /*! Request a transaction from Uniterm as documented in the Uniterm Guide.
126     *  The Key/Value pair params are a combination of the Parameters as
127     *  documented in the Uniterm Guide and the Monetra Client Interface
128     *  Protocol Spec.
129     * \param[in] uniterm_conn  Initialized connection to Uniterm
130     *                          as returned by uniterm_connect()
131     * \param[in] mparams      Array of key/value parameters to send to
132     *                         Uniterm
133     * \return Hashtable of string key/value pairs from response.  Please refer
134     *         to the Uniterm Guide and Monetra Client Interface Protocol
135     *         specification for the applicable list based on the action being
136     *         performed.  "code" and "u_errorcode" are always guaranteed to
137     *         be returned.
138     */
139    static Hashtable uniterm_sendrequest(Monetra uniterm_conn, Hashtable mparams)
140    {
141     int id;
142
143     Hashtable response = new Hashtable();
144
145     /* Request a new transaction from libmonetra */
146     id = uniterm_conn.TransNew();
147
148     /* For each item in the params hashtable, add it to the transaction */
149     foreach (DictionaryEntry kv in mparams) {
150      uniterm_conn.TransKeyVal(id, (String)kv.Key, (String)kv.Value);
151     }
152
153     /* Send the request to the Uniterm.  It will not return until
154      * a response is available, or a disconnect is detected */
155     if (!uniterm_conn.TransSend(id)) {
156      /* Disconnect detected, return an appropriate error condition!
157       * This should really never happen though... */
158      response["code"]        = "DENY";
159      response["u_errorcode"] = "CONN_ERROR";
160      response["verbiage"]    = "Connection to Uniterm failed: "
161                                + uniterm_conn.ConnectionError();
162      return response;
163     }
164
165     /* Save the response parameters from the Uniterm into a
166      * HashTable as our function prototype states. */
167     string[] keys = uniterm_conn.ResponseKeys(id);
```

```
168     for (int i=0; i < keys.Length; i++) {
169      response[keys[i]] = uniterm_conn.ResponseParam(id, keys[i]);
170     }
171
172     /* Free up some memory by purging unneeded data */
173     uniterm_conn.DeleteTrans(id);
174
175     return response;
176    }
177
178
179    /*! Tell Uniterm to shutdown.  Since we start it up, we should make sure
180     *  we turn it off prior to exiting otherwise the user will be prompted
181     *  with an error message stating the Uniterm is already running on the
182     *  next execution of this application!
183     * \param[in] uniterm_conn Initialized connection to the Uniterm
184     *                         as returned by uniterm_connect()
185     */
186    static void uniterm_shutdown(Monetra uniterm_conn)
187    {
188     Hashtable mparams = new Hashtable();
189     mparams["u_action"] = "shutdown";
190     uniterm_sendrequest(uniterm_conn, mparams);
191    }
192
193
194    /*! Main entry point to this application to be executed */
195    static void Main()
196    {
197     Monetra   uniterm_conn;
198     string    errorstr = "";
199     Hashtable response;
200
201     /* Step1: Launch the Uniterm */
202     uniterm_launch();
203     Console.WriteLine("Uniterm Launched");
204
205     /* Step2: Connect to the Uniterm */
206     uniterm_conn = uniterm_connect(ref errorstr);
207     if (uniterm_conn == null) {
208      Console.WriteLine("Failure: " + errorstr);
209      return;
210     }
211     Console.WriteLine("Connected to Uniterm");
212
213
214     /* Step3: Send txnrequest to Uniterm */
215     Hashtable mparams = new Hashtable();
216     /* Append the parameters for the txnrequest */
217     mparams["username"]     = monetra_user;
218     mparams["password"]     = monetra_pass;
219     mparams["u_action"]     = "txnrequest";
220     mparams["u_devicetype"] = "ingenico_rba";
221     mparams["u_device"]     = "HID";
222
223     /* Append the parameters for the transaction that will also get passed
224      * to Monetra such as the 'action', 'amount', etc. as described in the
225      * Monetra Client Interface Protocol Specification */
```

```
226    mparams["action"]   = "sale";
227    mparams["amount"]   = "12.00";
228    mparams["ordernum"] = "123456";
229    mparams["comments"] = "u_txnrequest";
230
231    response = uniterm_sendrequest(uniterm_conn, mparams);
232    if (String.Compare((string)response["code"], "AUTH", true) != 0) {
233     Console.WriteLine("Transaction failed.");
234    } else {
235     Console.WriteLine("Transaction SUCCESSFUL!");
236    }
237
238    /* Print out all the response key/value pairs ... */
239    foreach (DictionaryEntry kv in response) {
240     Console.WriteLine("\t" + (string)kv.Key + " = " + (string)kv.Value);
241    }
242
243    /* NOTE: No real reason to exit here ... we could just keep running
244     *       Step 3 all day long as long as you keep the uniterm_conn handle.
245     *       No reason to keep disconnecting and reconnecting, or
246     *       starting/stopping the Uniterm.
247     */
248
249    /* Step4: Cleanup */
250    uniterm_shutdown(uniterm_conn);
251
252    /* Connections will be automatically closed when the uniterm_conn
253     * initialized class is closed by the destructor/garbage
254     * collector */
255   }
256
257  }
258
259
```

## C.2 Microsoft C# using XML and `HttpWebRequest`

```
 1  /* UniTerm example program in C# using XML and HttpWebRequest
 2   *
 3   * Works with .Net Compact Framework v2
 4   *
 5   * Implemented based on the UniTerm Guide in conjunction with the
 6   * Monetra Client Interface Protocol Specification
 7   *
 8   * Please contact support@monetra.com with any questions
 9   */
10  using System;
11  using System.Diagnostics;
12  using System.Collections.Generic;
13  using System.Text;
14  using System.IO;
15  using System.Threading;
16  using System.Collections;
17  using System.Net;
18  using System.Xml;
19  using System.ComponentModel;
```

```
20  using System.Windows.Forms;
21  using System.Security.Cryptography.X509Certificates;
22
23  /* NOTE: if compiling with Mono, you can use
24   *        gmcs -r:System.Windows.Forms.dll utest_xml.cs
25   */
26
27  class utest_xml
28  {
29   /* Uniterm Connectivity Information
30   * NOTE: this is the default, it is possible to change, but 99%
31   *       of deployments will probably use this Uniterm information
32   *       as-is
33   */
34   private const string uniterm_host = "localhost";
35   private const int    uniterm_port = 8123;
36
37   /* Authentication information
38    * NOTE: This information corresponds with the public test server
39    *       at testbox.monetra.com:8665 */
40   private const string monetra_user = "test_retail:public";
41   private const string monetra_pass = "publ1ct3st";
42
43
44   static string uniterm_path()
45   {
46    switch (Environment.OSVersion.Platform) {
47     case PlatformID.Win32NT:
48     case PlatformID.Win32S:
49     case PlatformID.Win32Windows:
50     case PlatformID.WinCE:
51      return "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe";
52     default:
53      return "/usr/local/uniterm/bin/uniterm";
54    }
55   }
56
57
58   /*! Function to launch Uniterm from the current process.
59    *  If we don't launch it from the current process, it won't be given
60    *  focus! (at least on Windows this is true, until the first
61    *  manual focus is performed by an end-user) */
62   static void uniterm_launch()
63   {
64    Process uniterm = new Process();
65    uniterm.StartInfo.FileName = uniterm_path();
66    /* Not supported on CE
67     * uniterm.StartInfo.CreateNoWindow = true;
68     */
69
70    uniterm.Start();
71
72    /* Make sure Uniterm is ready before returning,
73    * Sleep 1000ms (1s) */
74    System.Threading.Thread.Sleep(1000);
75   }
76
77
```

```
 78  /*! Trust all SSL server certificates */
 79  internal class AcceptAllCertificatePolicy : ICertificatePolicy
 80  {
 81   public AcceptAllCertificatePolicy()
 82   {
 83   }
 84   public bool CheckValidationResult(ServicePoint sPoint,
 85                                     X509Certificate cert,
 86                                     WebRequest wRequest, int certProb)
 87   {
 88    // *** Always accept
 89    return true;
 90   }
 91  }
 92
 93
 94  /*! Function to POST and XML message to a Monetra-like entity
 95  *  (Monetra or Uniterm) via HTTPS.  It will return
 96  *  the key/value pairs from the XML response
 97  * \param[in]  host        Host to connect to
 98  * \param[in]  port        Port to connect to (via SSL/HTTPS)
 99  * \param[in]  xml         String-form XML to post
100  * \return True on successful communication, False if communication failed.
101  *         Note: True doesn't mean the transaction itself was successful.
102  */
103  static Hashtable uniterm_https_post(string host, int port, string xml)
104  {
105   Hashtable       response = new Hashtable();
106   string          url      = "https://" + host + ":" + port.ToString();
107   HttpWebRequest  req      = (HttpWebRequest)WebRequest.Create(url);
108   string          xmlout;
109
110   try {
111    /* POST Request */
112
113    /* Disable SSL Server Certificate Checking */
114    System.Net.ServicePointManager.CertificatePolicy =
115     new AcceptAllCertificatePolicy();
116
117    byte[] bytes;
118    bytes                 = System.Text.Encoding.ASCII.GetBytes(xml);
119    req.Method            = "POST";
120    req.ContentType       = "text/xml";
121    req.ContentLength     = bytes.Length;
122    Stream reqStream      = req.GetRequestStream();
123    reqStream.Write(bytes, 0, bytes.Length);
124    reqStream.Close();
125
126    /* Read Response */
127    /* Note issues with .Net CF v2 as per below:
128     *   http://blogs.msdn.com/b/andrewarnottms/archive/2007/11/19/why-net-compact-framework-fa
129     *   http://support.microsoft.com/kb/970549
130     * If the Server is OpenSSL, this can be worked around by setting
131     * SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS
132     */
133    HttpWebResponse resp   = (HttpWebResponse)req.GetResponse();
134    Stream respStream      = resp.GetResponseStream();
135    StreamReader rdr       = new StreamReader(respStream);
```

```
136     xmlout                = rdr.ReadToEnd();
137     rdr.Close();
138    } catch (System.Net.WebException e)  {
139     response["code"]        = "DENY";
140     response["u_errorcode"] = "CONN_ERROR";
141     response["verbiage"]    = "Connection to " + url + " failed: " +
142                               e.Message;
143     return response;
144    }
145    XmlDocument     xmldoc     = new XmlDocument();
146    xmldoc.LoadXml(xmlout);
147
148    XmlNodeList     trans      = xmldoc.DocumentElement.
149                                 SelectSingleNode("Resp").ChildNodes;
150    foreach (XmlNode kv in trans) {
151     response[kv.Name] = kv.InnerText;
152    }
153    return response;
154   }
155
156
157   /*! Request a ttransaction from Uniterm as documented in the Monetra
158    *  Uniterm Guide.  The Key/Value pair params are a combination of the
159    *  Parameters as documented in the Uniterm Guide and the Monetra Client
160    *  Interface Protocol Spec.
161    * \param[in] mparams   Array of key/value parameters to send to Uniterm
162    * \return Hashtable of string key/value pairs from response.  Please refer
163    *         to the Uniterm Guide and Monetra Client Interface Protocol
164    *         specification for the applicable list based on the action being
165    *         performed.  "code" and "u_errorcode" are always guaranteed to
166    *         be returned.
167    */
168   static Hashtable uniterm_sendrequest(Hashtable mparams)
169   {
170    string XML;
171
172    XML = "<MonetraTrans>" +
173           "<Trans identifier='1'>";
174
175    /* For each item in the params hashtable, add it to the transaction */
176    foreach (DictionaryEntry kv in mparams) {
177     XML = XML + "<" + (String)kv.Key + ">" + (String)kv.Value + "</" +
178           (string)kv.Key + ">";
179    }
180
181    XML = XML + "</Trans></MonetraTrans>";
182
183    return uniterm_https_post(uniterm_host, uniterm_port, XML);
184   }
185
186
187   /*! Tell Uniterm to shutdown.  Since we start it up,
188    *  we should make sure we turn it off prior to exiting otherwise
189    *  the user will be prompted with an error message stating
190    *  Uniterm is already running on the next execution
191    *  of this application!
192    */
193   static void uniterm_shutdown()
```

```
194   {
195    Hashtable mparams = new Hashtable();
196    mparams["u_action"] = "shutdown";
197    uniterm_sendrequest(mparams);
198   }
199
200
201   /*! Main entry point to this application to be executed */
202   static void Main()
203   {
204    Hashtable response;
205
206    /* Step1: Launch Uniterm */
207    uniterm_launch();
208    MessageBox.Show("Uniterm Launched");
209
210
211    /* Step2: Send txnrequest to Uniterm */
212    Hashtable mparams = new Hashtable();
213    /* Append the parameters for the ticket request as per the Monetra
214     * Uniterm Guide, section 4 */
215    mparams["username"]     = monetra_user;
216    mparams["password"]     = monetra_pass;
217    mparams["u_action"]     = "txnrequest";
218    mparams["u_devicetype"] = "ingenico_rba";
219    mparams["u_device"]     = "HID";
220
221    /* Append the parameters for the transaction that will also get passed
222     * to Monetra such as the 'action', 'amount', etc. as described in the
223     * Monetra Client Interface Protocol Specification */
224    mparams["action"]   = "sale";
225    mparams["amount"]   = "12.00";
226    mparams["ordernum"] = "123456";
227    mparams["comments"] = "u_txnrequest";
228
229    response = uniterm_sendrequest(mparams);
230    string resultMsg = "";
231    if (String.Compare((string)response["code"], "AUTH", true) != 0) {
232     resultMsg = "Transaction failed.\r\n";
233    } else {
234     resultMsg = "Transaction SUCCESSFUL!\r\n";
235    }
236
237    /* Print out all the response key/value pairs ... */
238    foreach (DictionaryEntry kv in response) {
239     resultMsg = resultMsg + (string)kv.Key + " = " + (string)kv.Value +
240                 "\r\n";
241    }
242
243    MessageBox.Show(resultMsg);
244
245    /* NOTE: No real reason to exit here ... we could just keep running
246     *       Step 2 all day long as long.
247     *       No reason to keep starting/stopping Uniterm.
248     */
249
250    /* Step3: Cleanup */
251    uniterm_shutdown();
```

```
252
253    /* Connections will be automatically closed when the uniterm_conn
254     * initialized class is closed by the destructor/garbage
255     * collector */
256   }
257  }
258
```

## C.3 Java using libmonetra

```
  1  /* Uniterm example program in Java
  2   *
  3   * Depends on the libmonetra Java native API
  4   *
  5   * Implemented based on the UniTerm Guide in conjunction with the
  6   * Monetra Client Interface Protocol Specification
  7   *
  8   * Please contact support@monetra.com with any questions
  9   */
 10  import java.util.Hashtable;
 11  import java.util.Enumeration;
 12  import com.mainstreetsoftworks.MONETRA;
 13
 14  /* Compile/run with:
 15   *   javac -classpath MONETRA.jar utest.java
 16   *   java -cp "./MONETRA.jar:." utest
 17   */
 18
 19  class utest {
 20   /* Uniterm Connectivity Information
 21    * NOTE: this is the default, it is possible to change, but 99%
 22    *       of deployments will probably use this uniterm information
 23    *       as-is
 24    */
 25   private static String uniterm_host     = "localhost";
 26   private static int    uniterm_port     = 8123;
 27
 28   /* Authentication information
 29    * NOTE: This information corresponds with the public test server
 30    *       at testbox.monetra.com:8665 */
 31   private static String monetra_user     = "test_retail:public";
 32   private static String monetra_pass     = "publ1ct3st";
 33
 34
 35   static String uniterm_path()
 36   {
 37    if (System.getProperty("os.name").startsWith("Windows")) {
 38     return "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe";
 39    } else {
 40     return "/usr/local/uniterm/bin/uniterm";
 41    }
 42   }
 43
 44
 45   /*! Function to launch Uniterm from the current process. If we don't
 46    *  launch it from the current process, it won't be given focus!
```

```
47     *  (at least on Windows this is true, until the first manual focus is
48     *  performed by an end-user) */
49    static void uniterm_launch()
50    {
51     try {
52      Process p = new ProcessBuilder(uniterm_path()).start();
53     } catch (java.io.IOException e) {
54      System.out.println(e.getMessage());
55      System.exit(1);
56     }
57     /* Make sure Uniterm is ready before returning,
58      * Sleep 1000ms (1s) */
59     try {
60      Thread.sleep(1000);
61     } catch (InterruptedException e) {
62     }
63    }
64
65
66    /*! Function to connect to an endpoint which uses the standard 'monetra'
67     * style protocol (so either Monetra itself, or Uniterm)
68     * \param[in]  host     Resolvable hostname or IP address to connect to
69     * \param[in]  port     Port associated with hostname to establish an SSL
70     *                      connection to
71     * \param[out] errorstr Textual error message if returns null
72     * \return Initialized connection class on success. null on failure
73     */
74    static MONETRA uniterm_connect_host(String host, int port,
75                                        StringBuilder errorstr)
76    {
77     /* Initialize the Class */
78     MONETRA conn = new MONETRA("");
79
80     errorstr.setLength(0);
81
82     /* We always want to use an SSL connection to Monetra and Uniterm */
83     conn.SetSSL(host, port);
84
85     /* Do not verify the SSL certificate, Monetra and Uniterm
86      * use self-signed certificates by default which cannot be validated.
87      * The connection is still encrypted, the endpoint just isn't strictly
88      * validated */
89     conn.VerifySSLCert(0);
90
91     /* This makes it so TransSend() will block until a response is
92      * received from Monetra.  Simplifies the API since we will never
93      * have more than one outstanding transaction per connection in
94      * this application */
95     conn.SetBlocking(1);
96
97     /* Connect! */
98     if (conn.Connect() == 0) {
99      errorstr.append(conn.ConnectionError());
100     return null;
101    }
102
103    return conn;
104   }
```

```
105
106
107   /*! Wrapper function to connect to Uniterm
108    * \param[out] errorstr Textual error message if returns null
109    * \return Initialized connection class on success. null on failure
110    */
111   static MONETRA uniterm_connect(StringBuilder errorstr)
112   {
113    MONETRA conn;
114    StringBuilder myerror = new StringBuilder();
115    conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror);
116    if (conn == null) {
117     errorstr.setLength(0);
118     errorstr.append("Connection to Uniterm Failed: " +
119                     myerror.toString());
120    }
121    return conn;
122   }
123
124
125   /*! Request a transaction from Uniterm as documented in the Monetra
126    *  Uniterm Guide.  The Key/Value pair params are a combination of the
127    *  Parameters as documented in the Uniterm Guide and the Monetra Client
128    *  Interface Protocol Spec.
129    * \param[in] uniterm_conn  Initialized connection to Uniterm
130    *                          as returned by uniterm_connect()
131    * \param[in] mparams       Array of key/value parameters to send to
132    *                          Uniterm
133    * \return Hashtable of string key/value pairs from response.  Please refer
134    *         to the Uniterm Guide and Monetra Client Interface Protocol
135    *         specification for the applicable list based on the action being
136    *         performed.  "code" and "u_errorcode" are always guaranteed to
137    *         be returned.
138    */
139   static Hashtable<String,String> uniterm_sendrequest(MONETRA uniterm_conn,
140          Hashtable<String,String> mparams)
141   {
142    long id;
143
144    Hashtable response = new Hashtable<String,String>();
145
146    /* Request a new transaction from libmonetra */
147    id = uniterm_conn.TransNew();
148
149    /* For each item in the params hashtable, add it to the transaction */
150    for (String key : mparams.keySet()) {
151     String value = mparams.get(key);
152     uniterm_conn.TransKeyVal(id, key, value);
153    }
154
155    /* Send the request to the Uniterm.  It will not return until
156     * a response is available, or a disconnect is detected */
157    if (uniterm_conn.TransSend(id) == 0) {
158     /* Disconnect detected, return an appropriate error condition!
159      * This should really never happen though... */
160     response.put("code",        "DENY");
161     response.put("u_errorcode", "CONN_ERROR");
162     response.put("verbiage",    "Connection to Uniterm failed:"
```

```
163                                 + uniterm_conn.ConnectionError());
164     return response;
165    }
166
167    /* Save the response parameters from the Uniterm into a
168     * HashTable as our function prototype states. */
169    String[] keys = uniterm_conn.ResponseKeys(id);
170    for (int i=0; i < keys.length; i++) {
171     response.put(keys[i], uniterm_conn.ResponseParam(id, keys[i]));
172    }
173
174    /* Free up some memory by purging unneeded data */
175    uniterm_conn.DeleteTrans(id);
176
177    return response;
178   }
179
180
181   /*! Tell Uniterm to shutdown.  Since we start it up,
182    *  we should make sure we turn it off prior to exiting otherwise
183    *  the user will be prompted with an error message stating the
184    *  Uniterm is already running on the next execution
185    *  of this application!
186    * \param[in] uniterm_conn Initialized connection to Uniterm
187    *                         as returned by uniterm_connect()
188    */
189   static void uniterm_shutdown(MONETRA uniterm_conn)
190   {
191    Hashtable mparams = new Hashtable<String,String>();
192    mparams.put("u_action", "shutdown");
193    uniterm_sendrequest(uniterm_conn, mparams);
194   }
195
196
197   /*! Main entry point to this application to be executed */
198   public static void main(String[] args)
199   {
200    MONETRA                  uniterm_conn;
201    StringBuilder            errorstr = new StringBuilder();
202    Hashtable<String,String> response;
203    String                   ticket;
204
205    /* Step1: Launch Uniterm */
206    uniterm_launch();
207    System.out.println("Uniterm Launched");
208
209    /* Step2: Connect to Uniterm */
210    uniterm_conn = uniterm_connect(errorstr);
211    if (uniterm_conn == null) {
212     System.out.println("Failure: " + errorstr.toString());
213     return;
214    }
215    System.out.println("Connected to Uniterm");
216
217    /* Step3: Send a txnrequest to Uniterm */
218    Hashtable<String,String> mparams = new Hashtable<String,String>();
219    /* Append the parameters for the txnrequest */
220    mparams.put("username",     monetra_user);
```

```
221    mparams.put("password",     monetra_pass);
222    mparams.put("u_action",     "txnrequest");
223
224    mparams.put("u_devicetype", "ingenico_rba");
225    mparams.put("u_device",     "HID");
226
227    /* Append the parameters for the transaction that will also get passed
228     * to Monetra such as the 'action', 'amount', etc. as described in the
229     * Monetra Client Interface Protocol Specification */
230    mparams.put("action",   "sale");
231    mparams.put("amount",   "12.00");
232    mparams.put("ordernum", "123456");
233    mparams.put("comments", "u_txnrequest");
234
235    response = uniterm_sendrequest(uniterm_conn, mparams);
236    if (!response.get("code").equalsIgnoreCase("AUTH")) {
237     System.out.println("Transasction failed.");
238    } else {
239     System.out.println("Transasction SUCCESSFUL!");
240    }
241
242    /* Print out all the response key/value pairs ... */
243    for (String key : response.keySet()) {
244     String value = response.get(key);
245     System.out.println("\t" + key + " = " + value);
246    }
247
248    /* NOTE: No real reason to exit here ... we could just keep running
249     *       Step 3 all day long as long as you keep the uniterm_conn handle.
250     *       No reason to keep disconnecting and reconnecting, or
251     *       starting/stopping the Uniterm.
252     */
253
254    /* Step4: Cleanup */
255    uniterm_shutdown(uniterm_conn);
256
257    /* Connections will be automatically closed when the uniterm_conn
258     * initialized classe is closed by the destructor/garbage
259     * collector */
260   }
261
262  }
263
264
```

## C.4 PHP using libmonetra

```
1  <?php
2  /* UniTerm example program in PHP
3   *
4   * Depends on the libmonetra PHP native API
5   *
6   * Implemented based on the UniTerm Guide in conjunction with the
7   * Monetra Client Interface Protocol Specification
8   *
9   * Please contact support@monetra.com with any questions
```

```
 */
error_reporting(E_ALL);
require_once("libmonetra.php");



/* Uniterm Connectivity Information
 * NOTE: this is the default, it is possible to change, but 99%
 *       of deployments will probably use this uniterm information
 *       as-is
 */
$uniterm_host         = "localhost";
$uniterm_port         = 8123;

/* Authentication information
 * NOTE: This information corresponds with the public test server
 *       at testbox.monetra.com:8665 */
$monetra_user     = "test_retail:public";
$monetra_pass     = "publ1ct3st";


/* Sets the path of the Uniterm executable.  Currently using
 * the default locations */
if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
 /* Windows path */
 $uniterm_path         = "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe";
} else {
 /* Unix path */
 $uniterm_path         = "/usr/local/uniterm/bin/uniterm";
}


/*! Function to launch Uniterm from the current process.
 *  If we don't launch it from the current process, it won't be given
 *  focus! (at least on Windows this is true, until the first
 *  manual focus is performed by an end-user) */
function uniterm_launch()
{
 global $uniterm_path;
 if (class_exists("COM")) {
  /* Must be running windows */
  $WshShell = new COM("WScript.Shell");
  $oExec    = $WshShell->Run('"' . $uniterm_path . '"', 10, false);
 } else {
  /* Must be on a Unix system */
  system("'" . $uniterm_path . "'" . " > /dev/null 2>&1 &");
 }

 /* Make sure Uniterm is ready before returning,
  * sleep 2s */
 sleep(2);
}


/*! Function to connect to an endpoint which uses the standard 'monetra'
 * style protocol (so either Monetra itself, or Uniterm)
 * \param[in]  host     Resolvable hostname or IP address to connect to
 * \param[in]  port     Port associated with hostname to establish an SSL
```

```
 68   *                       connection to
 69   * \param[out] errorstr Textual error message if returns null
 70   * \return Initialized connection on success. null on failure
 71   */
 72  function uniterm_connect_host($host, $port, &$errorstr)
 73  {
 74   /* Initialize the Connection */
 75   $conn = M_InitConn();
 76
 77   $errorstr = "";
 78
 79   /* We always want to use an SSL connection to Monetra and Uniterm */
 80   M_SetSSL($conn, $host, $port);
 81
 82   /* Do not verify the SSL certificate, Monetra and Uniterm
 83    * use self-signed certificates by default which cannot be validated.
 84    * The connection is still encrypted, the endpoint just isn't strictly
 85    * validated */
 86   M_VerifySSLCert($conn, false);
 87
 88   /* This makes it so TransSend() will block until a response is
 89    * received from Monetra.  Simplifies the API since we will never
 90    * have more than one outstanding transaction per connection in
 91    * this application */
 92   M_SetBlocking($conn, true);
 93
 94   /* Connect! */
 95   if (!M_Connect($conn)) {
 96    $errorstr = M_ConnectionError($conn);
 97    return null;
 98   }
 99
100   return $conn;
101  }
102
103
104  /*! Wrapper function to connect to Uniterm
105   * \param[out] errorstr Textual error message if returns null
106   * \return Initialized connection on success. null on failure
107   */
108  function uniterm_connect(&$errorstr)
109  {
110   global $uniterm_host, $uniterm_port;
111
112   $myerror = "";
113   $conn = uniterm_connect_host($uniterm_host, $uniterm_port, &$myerror);
114   if ($conn == null) {
115    $errorstr = "Connection to Uniterm Failed: " . $myerror;
116   }
117   return $conn;
118  }
119
120
121  /*! Request a transaction from Uniterm as documented in the Uniterm Guide.
122   *  The Key/Value pair params are a combination of the Parameters as
123   *  documented in the Uniterm Guide and the Monetra Client Interface Protocol
124   *  Spec.
125   * \param[in] uniterm_conn  Initialized connection to Uniterm as returned by
```

```
126   *                           uniterm_connect()
127   * \param[in] params         Array of key/value parameters to send to Uniterm
128   *
129   * \return Array of string key/value pairs from response.  Please refer to the
130   *         Uniterm Guide and Monetra Client Interface Protocol specification
131   *         for the applicable list based on the action being performed.
132   *         "code" and "u_errorcode" are always guaranteed to be returned.
133   */
134  function uniterm_sendrequest($uniterm_conn, $params)
135  {
136   $response = array();
137
138   /* Request a new transaction from libmonetra */
139   $id = M_TransNew($uniterm_conn);
140
141   /* For each item in the params array, add it to the transaction */
142   foreach ($params as $key => $value) {
143    M_TransKeyVal($uniterm_conn, $id, $key, $value);
144   }
145
146   /* Send the request to the Uniterm.  It will not return until a
147    * response is available, or a disconnect is detected */
148   if (!M_TransSend($uniterm_conn, $id)) {
149    /* Disconnect detected, return an appropriate error condition!
150     * This should really never happen though... */
151    $response["code"]        = "DENY";
152    $response["u_errorcode"] = "CONN_ERROR";
153    $response["verbiage"]    = "Connection to Uniterm failed: " .
154                               M_ConnectionError($uniterm_conn);
155    return $response;
156   }
157
158   /* Save the response parameters from the Uniterm into a HashTable
159    * as our function prototype states. */
160   $keys = M_ResponseKeys($uniterm_conn, $id);
161   foreach ($keys as $value) {
162    $response[$value]      = M_ResponseParam($uniterm_conn, $id, $value);
163   }
164
165   /* Free up some memory by purging unneeded data */
166   M_DeleteTrans($uniterm_conn, $id);
167
168   return $response;
169  }
170
171
172  /*! Tell Uniterm to shutdown.  Since we start it up,
173   *  we should make sure we turn it off prior to exiting otherwise
174   *  the user will be prompted with an error message stating the
175   *  Uniterm is already running on the next execution
176   *  of this application!
177   * \param[in] uniterm_conn Initialized connection to Uniterm
178   *                         as returned by uniterm_connect()
179   */
180  function uniterm_shutdown($uniterm_conn)
181  {
182   uniterm_sendrequest($uniterm_conn, array("u_action" => "shutdown"));
183  }
```

```
184
185
186
187  /* CODE TO EXECUTE ... */
188
189  $errorstr = "";
190
191  /* Step1: Launch Uniterm */
192  uniterm_launch();
193  echo "Uniterm Launched\r\n";
194
195  /* Step2: Connect to Uniterm */
196  $uniterm_conn = uniterm_connect(&$errorstr);
197  if ($uniterm_conn == null) {
198   echo "Failure: " . $errorstr . "\r\n";
199   return;
200  }
201
202  echo "Connected to Uniterm\r\n";
203
204
205  /* Step3: Send a txnrequest to the Uniterm */
206  $params  = array();
207
208  /* Append the parameters for the txnrequest */
209  $params["username"]     = $monetra_user;
210  $params["password"]     = $monetra_pass;
211  $params["u_action"]     = "txnrequest";
212  $params["u_devicetype"] = "ingenico_rba";
213  $params["u_device"]     = "HID";
214
215  /* Append the parameters for the transaction that will also get passed to
216   * Monetra such as the 'action', 'amount', etc. as described in the Monetra
217   * Client Interface Protocol Specification */
218  $params['action']   = 'sale';
219  $params['amount']   = '12.00';
220  $params['ordernum'] = '123456';
221  $params['comments'] = 'u_txnrequest';
222
223  $response = uniterm_sendrequest($uniterm_conn, $params);
224  if (strcasecmp($response["code"], "AUTH") != 0) {
225   echo "Transaction Failed.\r\n";
226  } else {
227   echo "Transaction SUCCESSFUL!\r\n";
228  }
229
230  /* Print out all the response key/value pairs ... */
231  foreach ($response as $key => $value) {
232   echo "\t" . $key . " = " . $value . "\r\n";
233  }
234
235  /* NOTE: No real reason to exit here ... we could just keep running
236   *       Step 3 all day long as long as you keep the uniterm_conn handle.
237   *       No reason to keep disconnecting and reconnecting, or
238   *       starting/stopping Uniterm.
239   */
240
241  /* Step4: Cleanup */
```

```
242  uniterm_shutdown($uniterm_conn);
243
244  /* Connections will be automatically closed when the uniterm_conn
245   * initialized connection is closed by the destructor/garbage collector */
246
247  ?>
248
249
```

## C.5 Microsoft VB.Net using libmonetra

```
 1  ' UniTerm example program in VB.Net
 2  '
 3  ' Depends on the libmonetra C# .Net native API (DLL)
 4  '
 5  ' Implemented based on the UniTerm Guide in conjunction with the
 6  ' Monetra Client Interface Protocol Specification
 7  '
 8  ' Please contact support@monetra.com with any questions
 9
10  Option Explicit On
11  Option Strict On
12
13  Imports System
14  Imports System.Collections
15  Imports System.Diagnostics
16  Imports System.Threading
17  Imports libmonetra
18
19  ' On unix, compile using:
20  '   gmcs /target:library /unsafe libmonetra.cs
21  '   vbnc2 -r:libmonetra.dll utest.vb
22
23  Module Module1
24   ' Uniterm Connectivity Information
25   ' NOTE: this is the default, it is possible to change, but 99%
26   '       of deployments will probably use this Uniterm information
27   '       as-is
28   Private Const uniterm_host As String = "localhost"
29   Private Const uniterm_port As Integer = 8123
30
31   ' Authentication information
32   '  NOTE: This information corresponds with the public test server
33   '        at testbox.monetra.com:8665
34   Private Const monetra_user As String = "test_retail:public"
35   Private Const monetra_pass As String = "publ1ct3st"
36
37   Private Function uniterm_path As String
38    Select Case Environment.OSVersion.Platform
39     Case PlatformID.Win32NT, PlatformID.Win32S, _
40          PlatformID.Win32Windows, PlatformID.WinCE
41      Return "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe"
42     Case Else
43      Return "/usr/local/uniterm/bin/uniterm"
44    End Select
45   End Function
```

```
 46
 47
 48   '! Function to launch Uniterm from the current process.
 49   '  If we don't launch it from the current process, it won't be given
 50   '  focus! (at least on Windows this is true, until the first
 51   '  manual focus is performed by an end-user)
 52   Private Sub uniterm_launch()
 53    Dim uniterm As New Process()
 54    uniterm.StartInfo.FileName = uniterm_path
 55    uniterm.StartInfo.CreateNoWindow = True
 56
 57    uniterm.Start()
 58
 59    ' Make sure Uniterm is ready before returning,
 60    ' Sleep 1000ms (1s)
 61    System.Threading.Thread.Sleep(1000)
 62   End Sub
 63
 64
 65   '! Function to connect to an endpoint which uses the standard 'monetra'
 66   '  style protocol (so either Monetra itself, or Uniterm)
 67   ' \param[in]  host     Resolvable hostname or IP address to connect to
 68   ' \param[in]  port     Port associated with hostname to establish an SSL
 69   '                      connection to
 70   ' \param[out] errorstr Textual error message if returns null
 71   ' \return Initialized connection class on success. null on failure
 72   Private Function uniterm_connect_host(ByVal host As String, ByVal port _
 73                                         As Integer, ByRef errorstr As String) _
 74                                         As Monetra
 75    ' Initialize the Class
 76    Dim conn As New Monetra
 77
 78    errorstr = ""
 79
 80    ' We always want to use an SSL connection to Monetra and Uniterm
 81    conn.SetSSL(host, port)
 82
 83    ' Do not verify the SSL certificate, Monetra and Uniterm
 84    ' use self-signed certificates by default which cannot be validated.
 85    ' The connection is still encrypted, the endpoint just isn't strictly
 86    ' validated
 87    conn.VerifySSLCert(False)
 88
 89    ' This makes it so TransSend() will block until a response is
 90    ' received from Monetra.  Simplifies the API since we will never
 91    ' have more than one outstanding transaction per connection in
 92    ' this application
 93    conn.SetBlocking(True)
 94
 95    ' Connect!
 96    If Not conn.Connect() Then
 97     errorstr = conn.ConnectionError()
 98     Return Nothing
 99    End If
100
101    Return conn
102   End Function
103
```

```
104
105   '! Wrapper function to connect to the Uniterm
106   ' \param[out] errorstr Textual error message if returns null
107   ' \return Initialized connection class on success. null on failure
108   Private Function uniterm_connect(ByRef errorstr As String) As Monetra
109    Dim conn As Monetra
110    Dim myerror As String = ""
111    conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror)
112    If conn Is Nothing Then
113     errorstr = "Connection to Uniterm Failed: " + myerror
114    End If
115
116    Return conn
117   End Function
118
119   ' Request a transaction from Uniterm as documented in the Uniterm Guide.
120   ' The Key/Value pair params are a combination of the Parameters as
121   ' documented in the Uniterm Guide and the Monetra Client Interface
122   ' Protocol Spec.
123   ' \param[in] uniterm_conn  Initialized connection to the Uniterm
124   '                         as returned by uniterm_connect()
125   ' \param[in] mparams      Array of key/value parameters to send to Uniterm
126   ' \return Hashtable of string key/value pairs from response.  Please refer
127   '         to the Uniterm Guide and Monetra Client Interface Protocol
128   '         specification for the applicable list based on the action being
129   '         performed.  "code" and "u_errorcode" are always guaranteed to
130   '         be returned.
131   Private Function uniterm_sendrequest(ByVal uniterm_conn As Monetra, ByVal _
132                                        mparams As Hashtable) As Hashtable
133    Dim id As Integer
134    Dim response As New Hashtable
135
136    ' Request a new transaction from libmonetra
137    id = uniterm_conn.TransNew()
138
139    ' For each item in the params hashtable, add it to the transaction
140    Dim kv As DictionaryEntry
141    For Each kv In mparams
142     uniterm_conn.TransKeyVal(id, CType(kv.Key, String), _
143                              CType(kv.Value, String))
144    Next kv
145
146    ' Send the request to the Uniterm.  It will not return until a
147    ' response is available, or a disconnect is detected
148    If Not uniterm_conn.TransSend(id) Then
149     ' Disconnect detected, return an appropriate error condition!
150     ' This should really never happen though...
151     response("code")        = "DENY"
152     response("u_errorcode") = "CONN_ERROR"
153     response("verbiage")    = "Connection to Uniterm failed:" _
154                               + uniterm_conn.ConnectionError()
155     Return response
156    End If
157
158    ' Save the response parameters from Uniterm into a
159    ' HashTable as our function prototype states. */
160    Dim keys() As String = uniterm_conn.ResponseKeys(id)
161    Dim i As Integer
```

```
 For i = 0 To keys.Length - 1
  response(keys(i)) = uniterm_conn.ResponseParam(id, keys(i))
 Next i

 ' Free up some memory by purging unneeded data
 uniterm_conn.DeleteTrans(id)

 Return response
End Function


'! Tell Uniterm to shutdown.  Since we start it up,
'  we should make sure we turn it off prior to exiting otherwise
'  the user will be prompted with an error message stating the
'  Uniterm is already running on the next execution
'  of this application!
' \param[in] uniterm_conn Initialized connection to Uniterm
'                         as returned by uniterm_connect()
Private Sub uniterm_shutdown(ByVal uniterm_conn As Monetra)
 Dim mparams As New Hashtable

 mparams("u_action") = "shutdown"
 uniterm_sendrequest(uniterm_conn, mparams)
End Sub

'! Main entry point to this application to be executed
Public Sub Main()
 Dim uniterm_conn As Monetra
 Dim errorstr As String = ""
 Dim response As Hashtable
 Dim ticket As String

 ' Step1: Launch Uniterm
 uniterm_launch()
 Console.WriteLine("Uniterm Launched")

 ' Step2: Connect to Uniterm
 uniterm_conn = uniterm_connect(errorstr)
 If uniterm_conn Is Nothing Then
  Console.WriteLine("Failure: " + errorstr)
  Return
 End If
 Console.WriteLine("Connected to Uniterm")

 ' Step3: Send a txnrequest to Uniterm
 Dim mparams As New Hashtable
 ' Append the parameters for the ticket request as per the Monetra
 ' Uniterm Guide
 mparams("username")       = monetra_user
 mparams("password")       = monetra_pass
 mparams("u_action")     = "txnrequest"
 mparams("u_devicetype") = "ingenico_rba"
 mparams("u_device")     = "HID"

 ' Append the parameters for the transaction that will also get passed
 ' to Monetra such as the 'action', 'amount', etc. as described in the
 ' Monetra Client Interface Protocol Specification
 mparams("action")   = "sale"
```

```
220    mparams("amount")   = "12.00"
221    mparams("ordernum") = "123456"
222    mparams("comments") = "u_txnrequest"
223
224    response = uniterm_sendrequest(uniterm_conn, mparams)
225    If StrComp(CType(response("code"), String), "AUTH", _
226                      vbTextCompare) <> 0 Then
227     Console.WriteLine("Transaction failed.")
228    Else
229     Console.WriteLine("Transaction SUCCESSFUL!")
230    End If
231
232    ' Print out all the response key/value pairs ...
233    Dim kv As DictionaryEntry
234    For Each kv In response
235     Console.WriteLine("  " + CType(kv.Key, String) + " = " + _
236                       CType(kv.Value, String))
237    Next kv
238
239    ' NOTE: No real reason to exit here ... we could just keep running
240    '       Step 3 all day long as long as you keep the uniterm_conn handle.
241    '       No reason to keep disconnecting and reconnecting, or
242    '       starting/stopping Uniterm.
243
244    ' Step4: Cleanup
245    uniterm_shutdown(uniterm_conn)
246
247    ' Connections will be automatically closed when the uniterm_conn
248    ' initialized class is closed by the destructor/garbage
249    ' collector
250   End Sub
251
252  End Module
253
254
```

## C.6 Microsoft `VBScript` using XML and `MSXML2`

```
  1  ' UniTerm example program in VBScript
  2  '
  3  ' Depends on the MSXML, and Microsoft Scripting Runtime
  4  '
  5  ' Implemented based on the UniTerm Guide in conjunction with the
  6  ' Monetra Client Interface Protocol Specification
  7  '
  8  ' Please contact support@monetra.com with any questions
  9
 10  Option Explicit
 11
 12  ' Monetra Connectivity Information
 13  Dim monetra_user
 14  Dim monetra_pass
 15
 16  ' Uniterm Connectivity Information
 17  Dim uniterm_host
 18  Dim uniterm_port
```

```
Dim uniterm_path


'! Function to launch Uniterm from the current process.
'  If we don't launch it from the current process, it won't be given
'  focus! (at least on Windows this is true, until the first
'  manual focus is performed by an end-user)
Sub uniterm_launch()
 Dim objShell
 Dim res
 Set objShell = CreateObject("Wscript.Shell")
 res = objShell.Run("""" & uniterm_path & """", 10, FALSE)

 ' Make sure Uniterm is ready before returning,
 ' Sleep 1000ms (1s)
 WScript.Sleep 1000
End Sub


'! Function to POST and XML message to a Monetra-like entity
'  (Monetra or the Uniterm) via HTTPS.  It will return
'  the key/value pairs from the XML response
'\param[in]  host        Host to connect to
'\param[in]  port        Port to connect to (via SSL/HTTPS)
'\param[in]  xml         String-form XML to post
'\param[out] errorstr    If returning False, the error message, typically comms
'                        error
'\param[out] myresponse  Dictionary of string key/value pairs from the response.
'\return True on successful communication, False if communication failed.
'        Note: True doesn't mean the transaction itself was successful.
Function uniterm_https_post(ByVal host, ByVal port, ByVal xml, ByRef errorstr, _
                        ByRef myresponse)
 Dim xmlhttp
 Dim xmldoc

 Set xmlhttp = CreateObject("MSXML2.ServerXMLHTTP")

 xmlhttp.open             "POST", "https://" & host & ":" & port, False
 xmlhttp.setOption        2, 13056
 ' Set Timeouts (in milliseconds)
 '   DNS: 5s, Connect: 5s, Send: 30s, Receive: 120s
 xmlhttp.setTimeouts      5000, 5000, 30000, 120000
 xmlhttp.setRequestHeader "Content-Type", "text/xml"

 On Error Resume Next
 xmlhttp.send             xml

 If Not Err.Number = 0 Then
  errorstr = "HTTPS POST Failed to https://" & host & ":" & port & _
             ": " & Err.Description
  uniterm_https_post = False
  Exit Function
 End If

 Set xmldoc   = CreateObject("Microsoft.XMLDOM")

 xmldoc.async = "false"
 xmldoc.loadxml(xmlhttp.responseText)
```

```
 77
 78   Dim Trans
 79   Set Trans = xmldoc.documentElement.selectSingleNode("Resp").childNodes
 80
 81   Dim kv
 82   For Each kv In Trans
 83    myresponse(kv.nodeName) = kv.text
 84   Next
 85
 86   uniterm_https_post = True
 87  End Function
 88
 89
 90  '! Request a transaction from Uniterm as documented in the Uniterm Guide.
 91  '  The Key/Value pair params are a combination of the Parameters as documented
 92  '  the Uniterm Guide and the Monetra Client Interface Protocol Spec.
 93  ' \param[in]  mparams    Dictionary of key/value parameters to send to the
 94  '                        Uniterm
 95  ' \param[out] errorstr   If returning False, the error message, typically comms
 96  '                        error
 97  ' \param[out] myresponse Dictionary of string key/value pairs from response.
 98  '                        Please refer to the Uniterm Guide and Monetra Client
 99  '                        Interface Protocol specification for the applicable
100  '                        list based on the action being performed.  "code" and
101  '                        "u_errorcode" are always guaranteed to be returned.
102  ' \return True on successful communication, False if communication failed.
103  '         Note: True doesn't mean the transaction itself was successful.
104  Function uniterm_sendrequest(ByVal mparams, ByRef errorstr, ByRef myresponse)
105   Dim xml
106
107   xml = "<MonetraTrans><Trans identifier='1'>"
108
109   ' For each item in the params dictionary, add it to the transaction
110   Dim key
111   For Each key In mparams
112    xml = xml & "<" & key & ">" & mparams(key) & "</" & key & ">"
113   Next
114
115   xml = xml & "</Trans></MonetraTrans>"
116
117   uniterm_sendrequest = uniterm_https_post(uniterm_host, uniterm_port, xml, _
118                                            errorstr, myresponse)
119  End Function
120
121
122  '! Tell Uniterm to shutdown.  Since we start it up,
123  '  we should make sure we turn it off prior to exiting otherwise
124  '  the user will be prompted with an error message stating the
125  '  Uniterm is already running on the next execution
126  '  of this application!
127  Sub uniterm_shutdown()
128   Dim myresponse
129   Dim errorstr
130   Dim mparams
131
132   Set mparams = CreateObject("Scripting.Dictionary")
133   mparams("u_action") = "shutdown"
134
```

```
 uniterm_sendrequest mparams, errorstr, myresponse
 ' No need for error checking in this function as we don't
 ' care if this fails
End Sub


'! Main entry point to this application to be executed

' Uniterm Connectivity Information
' NOTE: this is the default, it is possible to change, but 99%
'       of deployments will probably use this Uniterm information
'       as-is
uniterm_host = "localhost"
uniterm_port = 8123
uniterm_path = "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe"

' Authentication information
' NOTE: This information corresponds with the public test server
'       at testbox.monetra.com:8665
monetra_user = "test_retail:public"
monetra_pass = "publ1ct3st"


Dim errorstr
Dim mparams
Dim myresp
Dim msg

errorstr = ""

' Step1: Launch Uniterm
uniterm_launch
MsgBox("Uniterm Launched")


' Step2: Send txnrequest to Uniterm

Set myresp  = CreateObject("Scripting.Dictionary")
Set mparams = CreateObject("Scripting.Dictionary")
' Append the parameters for the txnrequest
mparams("username")     = monetra_user
mparams("password")     = monetra_pass
mparams("u_action")     = "txnrequest"
mparams("u_devicetype") = "ingenico_rba"
mparams("u_device")     = "HID"

' Append the parameters for the transaction that will also get passed
' to Monetra such as the 'action', 'amount', etc. as described in the
' Monetra Client Interface Protocol Specification
mparams("action")   = "sale"
mparams("amount")   = "12.00"
mparams("ordernum") = "123456"
mparams("comments") = "u_txnrequest"

If Not uniterm_sendrequest(mparams, errorstr, myresp) Then
 MsgBox errorstr
 WScript.Quit 1
End If
```

```
193
194  If StrComp(myresp("code"), "AUTH", vbTextCompare) <> 0 Then
195   msg = "Transaction failed." & vbNewLine
196  Else
197   msg = "Transaction SUCCESSFUL!" & vbNewLine
198  End If
199
200  ' Print out all the response key/value pairs ...
201  Dim key
202  For Each key In myresp
203   msg = msg & "  " & key & " = " & myresp(key) & vbNewLine
204  Next
205
206  MsgBox (msg)
207
208
209  ' NOTE: No real reason to exit here ... we could just keep running
210  '       Step 2 all day long.  No reason to keep starting/stopping the
211  '       Uniterm.
212
213  ' Step3: Cleanup
214  uniterm_shutdown
215
216
217
218
219
```

## C.7 Microsoft Visual Basic 6 using libmonetra

```
 1  Attribute VB_Name = "Module1"
 2  ' UniTerm example program in VB6
 3  '
 4  ' Depends on the libmonetra C# .Net native API (DLL) (has COM hooks)
 5  '
 6  ' Must add reference to libmonetra and Microsoft Scripting Runtime
 7  '
 8  ' Implemented based on the UniTerm Guide in conjunction with the
 9  ' Monetra Client Interface Protocol Specification
10  '
11  ' Please contact support@monetra.com with any questions
12
13  Option Explicit
14
15  ' MonetraInformation
16  Dim monetra_user As String
17  Dim monetra_pass As String
18
19  ' Uniterm Connectivity Information
20  Dim uniterm_host As String
21  Dim uniterm_port As Integer
22  Dim uniterm_path As String
23
24  Private Declare Sub Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)
25
26  '! Function to launch Uniterm from the current process.
```

```
'  If we don't launch it from the current process, it won't be given
'  focus! (at least on Windows this is true, until the first
'  manual focus is performed by an end-user)
Sub uniterm_launch()
 Dim id As Double
 id = Shell("""" & uniterm_path & """", vbNormalFocus)

 ' Make sure Uniterm is ready before returning,
 ' Sleep 1000ms (1s)
 Sleep (1000)
End Sub


'! Function to connect to an endpoint which uses the standard 'monetra'
' style protocol (so either Monetra itself, or Uniterm)
' \param[in]  host     Resolvable hostname or IP address to connect to
' \param[in]  port     Port associated with hostname to establish an SSL
'        connection to
' \param[out] errorstr Textual error message if returns null
' \return Initialized connection class on success. null on failure
Function uniterm_connect_host(ByVal host As String, ByVal port As Integer, _
    ByRef errorstr As String) As IMonetra
 ' Initialize the Class
 Dim conn As IMonetra
 Set conn = New Monetra

 errorstr = ""

 ' We always want to use an SSL connection to Monetra and Uniterm
 conn.SetSSL host, port

 ' Do not verify the SSL certificate, Monetra and Uniterm
 ' use self-signed certificates by default which cannot be validated.
 ' The connection is still encrypted, the endpoint just isn't strictly
 ' validated
 conn.VerifySSLCert False

 ' This makes it so TransSend() will block until a response is
 ' received from Monetra.  Simplifies the API since we will never
 ' have more than one outstanding transaction per connection in
 ' this application
 conn.SetBlocking True

 ' Connect!
 If Not conn.Connect() Then
  errorstr = conn.ConnectionError()
  Set uniterm_connect_host = Nothing
  Exit Function
 End If

 Set uniterm_connect_host = conn
End Function


'! Wrapper function to connect to Uniterm
' \param[out] errorstr Textual error message if returns null
' \return Initialized connection class on success. null on failure
Function uniterm_connect(ByRef errorstr As String) As IMonetra
```

```
 85   Dim conn As IMonetra
 86   Dim myerror As String
 87
 88   myerror = ""
 89   Set conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror)
 90   If conn Is Nothing Then
 91    errorstr = "Connection to Uniterm Failed: " & myerror
 92   End If
 93   Set uniterm_connect = conn
 94  End Function
 95
 96
 97  ' Request a transaction from Uniterm as documented in the UniTerm
 98  ' Guide.  The Key/Value pair params are a combination of the Parameters as
 99  ' Uniterm Guide and the Monetra Client Interface Protocol Spec.
100  ' \param[in] uniterm_conn  Initialized connection to Unitermas returned by
101  '                         connect_to_uniterm()
102  ' \param[in] mparams       Dictionary of key/value parameters to send to
103  '                         Uniterm
104  ' \return Dictionary of string key/value pairs from response.  Please refer
105  '  to the Uniterm Guide and Monetra Client Interface Protocol
106  '  specification for the applicable list based on the action being
107  '  performed.  "code" and "u_errorcode" are always guaranteed to
108  '  be returned.
109  Function uniterm_sendrequest(ByVal uniterm_conn As IMonetra, _
110      ByVal mparams As Dictionary) _
111      As Dictionary
112   Dim id As Integer
113   Dim myresponse As New Dictionary
114
115   ' Request a new transaction from libmonetra
116   id = uniterm_conn.TransNew()
117
118   ' For each item in the params dictionary, add it to the transaction
119   Dim key
120   For Each key In mparams
121    uniterm_conn.TransKeyVal id, key, mparams(key)
122   Next key
123
124   ' Send the request to the Uniterm.  It will not return until a
125   ' response is available, or a disconnect is detected
126   If Not uniterm_conn.TransSend(id) Then
127    ' Disconnect detected, return an appropriate error condition!
128    ' This should really never happen though...
129    myresponse("code") = "DENY"
130    myresponse("u_errorcode") = "CONN_ERROR"
131    myresponse("verbiage") = "Connection to Uniterm failed: " _
132        & uniterm_conn.ConnectionError()
133    Set uniterm_sendrequest = myresponse
134    Exit Function
135   End If
136
137   ' Save the response parameters from the Uniterm into a HashTable
138   ' as our function prototype states.
139   Dim keys() As String
140   keys = uniterm_conn.ResponseKeys(id)
141   Dim i As Integer
142   For i = LBound(keys) To UBound(keys)
```

```
143    myresponse(keys(i)) = uniterm_conn.ResponseParam(id, keys(i))
144   Next i
145
146   ' Free up some memory by purging unneeded data
147   uniterm_conn.DeleteTrans (id)
148
149   Set uniterm_sendrequest = myresponse
150  End Function
151
152
153  '! Tell Uniterm to shutdown.  Since we start it up, we should make sure we
154  '  turn it off prior to exiting otherwise the user will be prompted with an
155  '  error message stating Uniterm is already running on the next execution
156  '  of this application!
157  ' \param[in] uniterm_conn Initialized connection to Uniterm as returned by
158  '            connect_to_uniterm()
159  Sub uniterm_shutdown(ByVal uniterm_conn As IMonetra)
160   Dim mparams As New Dictionary
161
162   mparams("u_action") = "shutdown"
163   uniterm_sendrequest uniterm_conn, mparams
164  End Sub
165
166
167  '! Main entry point to this application to be executed
168  Sub Main()
169   ' Uniterm Connectivity Information
170   ' NOTE: this is the default, it is possible to change, but 99%
171   '       of deployments will probably use this Uniterm information
172   '       as-is
173   uniterm_host = "localhost"
174   uniterm_port = 8123
175   uniterm_path = "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe"
176
177   ' Authentication information
178   ' NOTE: This information corresponds with the public test server
179   '       at testbox.monetra.com:8665
180   monetra_user = "test_retail:public"
181   monetra_pass = "publ1ct3st"
182
183   Dim uniterm_conn As IMonetra
184   Dim errorstr As String
185   Dim myresp As Dictionary
186   Dim msg As String
187
188   errorstr = ""
189
190   ' Step1: Launch Uniterm
191   uniterm_launch
192   MsgBox ("Uniterm Launched")
193
194   ' Step2: Connect to Uniterm
195   Set uniterm_conn = uniterm_connect(errorstr)
196   If uniterm_conn Is Nothing Then
197    MsgBox ("Failure: " & errorstr)
198    Exit Sub
199   End If
200
```

```
 MsgBox ("Connected to the Uniterm")

 ' Step3: Send a txnrequest to Uniterm
 Dim mparams As New Dictionary
 ' Append the parameters for the ticket request as per the Uniterm Guide
 mparams("username")     = monetra_user
 mparams("password")     = monetra_pass
 mparams("u_action")     = "txnrequest"
 mparams("u_devicetype") = "ingenico_rba"
 mparams("u_device")     = "HID"

 ' Append the parameters for the transaction that will also get passed
 ' to Monetra such as the 'action', 'amount', etc. as described in the
 ' Monetra Client Interface Protocol Specification
 mparams("action")       = "sale"
 mparams("amount")       = "12.00"
 mparams("ordernum")     = "123456"
 mparams("comments")     = "u_txnrequest"

 Set myresp = uniterm_sendrequest(uniterm_conn, mparams)
 If StrComp(myresp("code"), "AUTH", vbTextCompare) <> 0 Then
  msg = "Transaction failed." & vbNewLine
 Else
  msg = "Transaction SUCCESSFUL!" & vbNewLine
 End If

 ' Print out all the response key/value pairs ...
 Dim key
 For Each key In myresp
  msg = msg & "  " & key & " = " & myresp(key) & vbNewLine
 Next key
 MsgBox (msg)

 ' NOTE: No real reason to exit here ... we could just keep running
 '       Step 3 all day long as long as you keep the uniterm_conn handle.
 '       No reason to keep disconnecting and reconnecting, or
 '       starting/stopping Uniterm.

 ' Step4: Cleanup
 uniterm_shutdown uniterm_conn

 ' Connections will be automatically closed when the uniterm_conn initialized
 ' class is cleaned up by the destructor/garbage collector
End Sub
```

# D PCI Security and Implementation

The below details the various security and PCI requirements and how deployments may be impacted. Integrators and distributors should read this section prior to any production deployments.

UniTerm depends on an instance of Monetra v8.y.z being accessible. Monetra may run on customer-hosted equipment or be provided as a service or gateway via a third party.

| TOPIC | DISCUSSION |
|---|---|
| Delete sensitive authentication data stored by previous payment application versions. | UniTerm has never stored any sensitive authentication data. |
| Delete any sensitive authentication data (pre-authorization) gathered as a result of troubleshooting the payment application. | UniTerm has never stored any sensitive authentication data, even for troubleshooting purposes. |
| Securely delete cardholder data after customer-defined retention period. | UniTerm never stores cardholder data. |
| Mask PAN when displayed so only personnel with a business need can see the full PAN. | UniTerm mandates the use of users with the `obscured` flag, therefore it is not possible that the full PAN can ever be returned. |
| Render PAN unreadable anywhere it is stored (including data on portable digital media, backup media, and in logs). | UniTerm never stores cardholder data, nor does it have its own logging facilities. |
| Protect keys used to secure cardholder data against disclosure and misuse. | UniTerm never stores cardholder data and therefore does not utilize keys. |
| Implement key-management processes and procedures for cryptographic keys used for encryption of cardholder data. | UniTerm never stores cardholder data and therefore does not utilize keys. |
| Implement secure key-management functions. | UniTerm never stores cardholder data and therefore does not utilize keys. |
| Provide a mechanism to render irretrievable cryptographic key material or cryptograms stored by the payment application. | UniTerm never stores cardholder data and therefore does not utilize keys. |

| | |
|---|---|
| Use unique user IDs and secure authentication for administrative access and access to cardholder data. | UniTerm does not provide or facilitate administrative access. |
| Use unique user IDs and secure authentication for access to PCs, servers, and databases with payment applications. | UniTerm does not provide or facilitate administrative or remote access.<br><br>Use unique user names and secure authentication to access any PCs, servers, and databases with payment applications and/or cardholder data. |
| Implement automated audit trails. | UniTerm does not provide its own logging facilities, instead all requests are sent to Monetra which logs and maintains the audit trails on behalf of UniTerm. |
| Facilitate centralized logging. | Since UniTerm sends all requests and metadata to Monetra, Monetra is responsible for facilitating centralized logging. |
| Implement and communicate application versioning methodology. | Please see the Versioning section. |
| Securely implement wireless technology. | UniTerm is not designed to use or facilitate the use of wireless technologies. |
| Secure transmissions of cardholder data over wireless networks. | UniTerm is not designed to use or facilitate the use of wireless technologies. |
| Provide instructions for secure use of wireless technology. | Integrators should ensure they secure any wireless technologies in use in compliance with the requirements in PA-DSS Requirement 6.3 |
| Use only necessary and secure services, protocols, components and dependent software and hardware, including those provided by third parties. | UniTerm communicates only via TLS using proprietary protocols to a Monetra server (hosted or customer-owned) across an intranet or the Internet. A customer may choose to deploy UniTerm with one or more hardware card entry devices or terminals directly attached to the System via USB, Serial, BlueTooth, or Ethernet<br><br>Integrators must ensure only necessary and secure protocols, services, etc., are used on the system. |
| Store cardholder data only on servers not connected to the Internet | UniTerm never stores cardholder data. |
| Implement two-factor authentication for all remote access to payment application that originates from outside the customer environment. | UniTerm never stores cardholder data, nor does it provide access to cardholder data. Integrators must ensure that all remote access originating from outside the customer's network to a payment application (Monetra) must use two-factor authentication. |
| Securely deliver remote payment application updates. | Integrators must securely deliver updates to UniTerm in compliance with the Deployment section. Deployments must |

| | |
|---|---|
| | be done in accordance with the PCI PA-DSS requirement 10.3. |
| Securely implement remote-access software. | Main Street Softworks will never reach out to a remote customer network. If an Integrator chooses to support remote access for management they must do so in compliance with PA-DSS Requirement 10.3.2. |
| Secure transmissions of cardholder data over public networks. | UniTerm communicates only via TLS to Monetra using proprietary protocols.<br><br>UniTerm communicates using the PCI DSS required protocols and cipher suites automatically (`TLSv1.2 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`). No configuration is required of UniTerm to comply, nor is it possible to reconfigure UniTerm to a less secure protocol or cipher suite. Future versions may implement new protocols and cipher suites as they become available. |
| Encrypt cardholder data sent over end-user messaging technologies. | UniTerm does not facilitate or support the use of end-user messaging technologies. |
| Encrypt non-console administrative access. | UniTerm does not provide or facilitate administrative access. |