UniTerm® Payment Interface Application

UniTerm Integration and Deployment Guide

Revision: 8.0
Publication date August 17, 2015

UniTerm Integration and Deployment Guide

Main Street Softworks, Inc.

Revision: 8.0

Publication date August 17, 2015

Legal Notice

Table of Contents

1 Revision History

Version	Date	Changes
v8.0.0	2015-08-17	• Initial revision

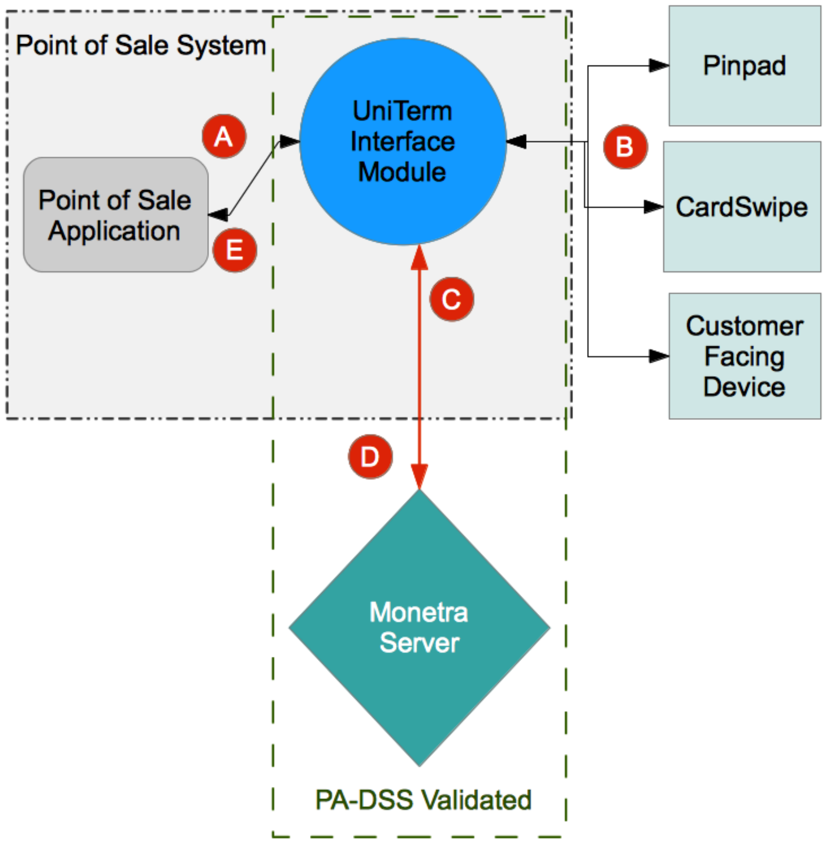
2 UniTerm System

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. If UniTerm was designed as a library rather than a separate application, it would be considered part of the integrator's application. This would mandate that integrators validate to PCI PA-DSS as well as go through direct end-to-end EMV brand certifications.

UniTerm's goal is to assist integrators in avoiding the costly and time consuming validations and certifications. While it would have been easier to implement as a library, and easier for integrators to use if it was a library, we strongly believe that the benefits far outweigh the integration and deployment inconveniences.

3 UniTerm Integration and Deployment Overview

3.1 Deployment

For Desktop deployments, Uniterm should be bundled and distributed with the POS system.

Provided to each integrator is a license for UniTerm that is installed via the Monetra Installer, however, production deployments will not use this distribution mechanism. Instead, integrators should package the directory that is created after installation to distribute with their own package (e.g. POS). 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.

For Android deployments, Uniterm is available as a standalone `APK` package. It can also be bundled with the POS system, please contact us for guidance on 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, or even simply used for Clerk status feedback, will also consume a UniTerm license.

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.

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

3.7.1 Location

The location of the `uniterm.ini` 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 `monetra_uniterm.exe` executable
- Mac OS X:
  - `~/Library/Application Support/Uniterm/uniterm.ini`
- Linux/Unix:
  - `~/.uniterm/uniterm.ini`
  - same path as the `monetra_uniterm` executable

Note: If the `uniterm.ini` 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`".

Under the `[monetra]` section:

- `host`: Required. Hostname/address where Monetra resides
- `port`: Required. Port to connect to Monetra on

Under the `[uniterm]` section:

- `port`: Required if using SSL. 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`
- `ssl_enabed`: Optional. Can only be disabled on Android (where the default is `no`), other systems SSL is always enabled

- `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.
- `nosigfloor`: Optional. If not specified, defaults to disabled, should be specified as a dollar amount. This configuration value is a temporary stop-gap until Monetra supports advertising a merchant's desired floor limit for requiring signatures and will be removed in the future. This only applies to Swiped transactions as EMV follows chip-specific rules. 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.

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, except on Android where you have the option to use Service-based communication (network-communication is available as a configurable option).

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 (127.0.0.1) via IPv4 on port 8123. It is recommended to use the ip address rather than 'localhost' since some operating systems may not fall back to trying IPv4. However, 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 shared secret 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 for an example of how to utilize this communication option.

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 Required User Permissions

The Monetra user identified by the username must be a Monetra SUB-USER with the `obscure_sensitive_data` flag set. For pass-through operations the subuser must have additional permissions for the transaction types it will perform (such as 'sale').

Uniterm also requires these permissions to operate:

- `CHKPWD`
- `MERCHINFO`
- `GETPERMS`
- `SALE`
- `VOID`
- `REVERSAL`
- `IMAGEADD`
- `TERMLOAD` - EMV or INTERAC ONLY
- `EMVCOMPLETE` - EMV ONLY
- `INTERACMAC` - Canadian Interac Debit Only

More permissions may be required based on the POS operations supported.

4 UniTerm Protocol

4.1 Overview

Application software communicates with the UniTerm module via the UniTerm protocol (which is similar to the Monetra Client Interface Protocol).

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`	`DEVICELOAD`. 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. If the load for the device is identical to the previous load, the load will be skipped. 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. If the device or merchant account does not support EMV or Interac, this command will simply return success. Optionally `u_deviceidlemessage` may be passed to this as well to set the device's default message if supported by the device. `TXNREQUEST` Transaction Request. Sensitive data (trackdata, account, cvv, avs, pin) will be retrieved by the UniTerm module either via the GUI or via a card entry device and forwarded to the Monetra server. `CARDREQUEST` 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` This action performs a direct pass-through of parameters to the Monetra server. Only the username, password, and u_action parameters are required. Can be used for reports, etc.

`CANCEL` Will attempt to cancel an outstanding `TXNREQUEST`. 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`.

`DEVICETYPES` Will return a comma separated list of device types supported by the UniTerm module. No authentication required.

Headers:
- `devicetype` - internal device name
- `manufacturer` - textual description of device manufacturer
- `model` - textual description of model
- `connectivity` - pipe separated list of connectivity methods supported by the device, e.g. `SERIAL|USB|BLUETOOTH|IP`
- `functionality` - pipe separated list of functionality supported by the device:

  e.g. SIGNATURE|SWIPE|RESET|IDLE|REQKEY|REQPIN

`STATUS` Requests the current status of a pending `TXNREQUEST`. Requires 'username', 'password' and 'u_id' fields which must match the pending request. This will provide a textual verbiage response suitable for clerk display.

`SERIALLIST` Will return a comma separated list of all serial ports enumerated on the system. No authentication required.

Headers:
- `port` - The port path
- `desc` - Description of port, if applicable

	`BLUETOOTHLIST` 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. Headers: • `name` - The textual name of the device as registered with the operating system. • `mac` - The device bluetooth `MAC` address • `uuid` - The device bluetooth `UUID` `USBLIST` List all supported USB devices that are currently attached to the machine UniTerm is running on. Headers: • `nickname` - The devicetype (device internal name) associated with the USB entry • `devpath` - The device path of the USB device (windows only) • `bus` - The USB bus (linux only) • `addr` - The USB address (linux only) • `vendorID` - The USB vendor id • `productID` - The USB product id • `serial_number` - The device serial number (if provided) `SHUTDOWN` Terminates execution of the UniTerm module. This should be called when the application software terminates. `MODIFYCONFIG` 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` 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.
`u_flags`	`Txnrequest` and `Cardrequest` only. Multiple flags may be sent per data ticket request. All flags are separated by a pipe (|) symbol. • `ENCRYPTEDONLY` - Permit encrypted reader data only. Not valid on `cardrequest` • `DEVICEONLY` - Suppresses display of clerk facing dialog. For instance, on a swipe request, no swipe dialog would be presented. Important note: keyboard emulation readers

	are not supported with this flag, only serial, USB HID, and Bluetooth devices can work. On Android, this flag is automatically implied due to the fact that it does not support a GUI mode of operation. • KEY - Perform capture of manually keyed data. Not valid on cardrequest • AVS - Request AVS data. Only allowed on keyed transactions. • CVV - Request for CVV data. Only allowed on keyed transactions.
u_cardclass	Txnrequest 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 • DEBIT - DEBIT card transaction • EBT - Electronic Benefits transaction • INTERAC - Canadian Interac card transaction • GIFT - Gift card transaction
u_device	Txnrequest, Cardrequest, and DeviceLoad only. 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. • USB • SER:port - Serial • BT:mac,[uuid] - Bluetooth • IP:ipaddr:port - IP
u_devicetype	Txnrequest, Cardrequest, and DeviceLoad only. The unique device type supported by Uniterm as found via a devicelist request. Required if u_device is provided.
u_language	Txnrequest only. Used to override terminal defaults for display of text prompts. Current choices are "en" or fr".
u_currency	Txnrequest 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 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`, `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` only. Transaction amount. Required.
`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.

4.3 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 sig 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 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 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.

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.
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.

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 sent 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 NOT apply to the US market.

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)
- http://usa.visa.com/download/merchants/play-it-smart-with-chip-payment-transactions.pdf (page 3)

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

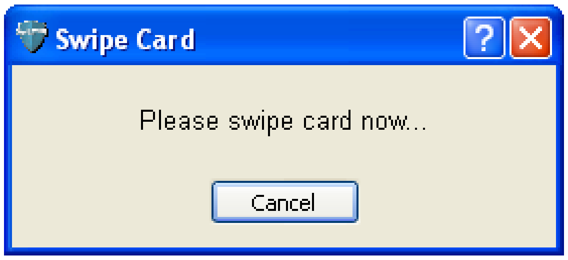
6.3 Pin Debit (forced) Transaction Request

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



Please swipe card now...

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 `monetra_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 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	NONE	x
iUP250 uCPX	uCPX	ingenico_ucpx	Canada	NONE	x
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 for running EMV transactions. Any device in the Ingenico RBA family may be used.

The RBA family includes all Ingenico Telium2 devices that can run the RBA (Retail Base Application) software version 14 or higher. This includes, but is not limited to:

- `iCMP`
- `iPP320`
- `iPP350`
- `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->Serial` - Requires `Telium` or `Jungo` drivers on Windows, will show up as a virtual COM port.
- `Serial - 115200 8N1 - No flow control`
- `Bluetooth` - Android only
- `IP - server mode`

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.

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`.

9.1.1.3 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 an `RBA12` or `RBA13` firmware release. 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 in order to obtain the necessary form building tools, and information on how to upload custom forms and images onto the device. Integrators should also contact their device distributor to ask about services to pre-load any files prior to shipping devices to customers. 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`. 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.

- 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.

9.1.2 Verifone `VX` `XPI` information

The minimum 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.

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.
- `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.

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	Paymentech Tampa 3.2.0 (Jan 2015)	Canada, Attended, OfflinePin, Sig
Verifone vx805	6.2.0/1C	8.0	Paymentech Tampa 3.2.0 (Oct 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Moneris				
Ingenico iPP320 CPX	4.66/3C	8.0	Moneris SPDH 2.0.0 (Sep 2015)	Canada, Attended, OfflinePin, Sig
Ingenico iUN uCPX	4.66/15C	8.0	Moneris SPDH 2.0.0 (Sep 2015)	Canada, Unattended, OfflinePin, NoSig
First Data				
Ingenico RBA family	4.67/1C	8.0	First Data Cardnet or Nashville EDC 4.1.0 (Oct 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Verifone vx805	6.2.0/1C	8.0	First Data Cardnet or Nashville EDC 4.1.0	USA, Attended, OnlinePin, OfflinePin, Sig

Device	EMVKern/Conf	UniTerm	Module Version	Config
			(Nov 2015)	
Tsys				
Ingenico RBA family	4.67/1C	8.0	TSYS (aka Vital/VisaNet) 3.0.0 (Nov 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Verifone vx805	6.2.0/1C	8.0	TSYS (aka Vital/VisaNet) 3.0.0 (Nov 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Global Payments				
Ingenico RBA family	4.67/1C	8.0	Global Payments East 3.0.0 (Oct 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Verifone vx805	6.2.0/1C	8.0	Global Payments East 3.0.0 (Oct 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
WorldPay				
Ingenico RBA family	4.67/1C	8.0	RBS WorldPay TCMP 2.0.0 (Oct 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Vantiv				
Ingenico RBA family	4.67/1C	8.0	Vantiv/ FifthThird 610 2.1.0 (Dec 2015)	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

Key	Description
`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. 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 ever 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.
- Signature line (if necessary) - `u_need_signature=yes`
- 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: • `G:` Keyed entry (EMV Fallback) • `M:` Keyed entry • `T:` EMV Contactless • `C:` EMV Contact • `F:` Swipe (EMV Fallback) • `R:` `MSD (RFID)` Contactless

	• `S:` Swipe • `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: • `none` • `sig` • `pin` • `pinsig` • `unknown` 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. • `AAC` - Application Authentication Cryptogram (decline) • `ARQC` - Application Request Cryptogram (intermediate or contactless) • `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. • `Tax` - Tax amount for order • `Amount` - Authorized amount, either the amount passed in or the partially approved amount provided in the `authamount` field. • `Cash Back Amount` - Amount of Cash Back (currently not supported)
`Card Disposition`	• When `u_errorcode` is `EMV_CARD_REMOVED`, should say: "CARD REMOVED" • When `u_errorcode` is `EMV_CARD_DENY`, should say: "DECLINED BY CARD" • When `rcpt_emv_cvm` is `pin` or `pinsig`, should say: "VERIFIED BY PIN" • When `rcpt_entry_mode` is `F` or `G`, should say: "CHIP CARD SWIPED" • When `authamount` is returned and is not equal to requested amount, should say: "TRANSACTION PARTIALLY APPROVED"

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	XXXXXXXXXXX2513
auth	196805
batch	1
cardholdername	AEIPS 24 TEST CARD
cardtype	AMEX
code	AUTH
item	570
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_custom	refnum:660136000010015700
rcpt_emv_ac	5C221DC28EB72FCF
rcpt_emv_actype	TC
rcpt_emv_aid	A000000025010801
rcpt_emv_cvm	sig
rcpt_emv_name	AMERICAN EXPRESS
rcpt_emv_tsi	FC00
rcpt_emv_tvr	0000008000
rcpt_entry_mode	C
rcpt_issuer_resp_code	00
rcpt_resp_code	025
timestamp	1437407566
ttid	743
u_errorcode	SUCCESS
u_need_signature	yes

B.7.1.2 Example Receipt

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

          SALE

MID: 1625         Lane: 1
AMEX                    C
Card:     XXXXXXXXXXX2513
Date/Time:   072015115246
Order #:             5096
TTID:                 743
Auth: 196805     Batch: 1
refnum:660136000010015700

AMOUNT:             34.00

        APPROVED

SIGNATURE


______________
```

```

CARDHOLDER WILL PAY CARD
   ISSUER ABOVE AMOUNT
 PURSUANT TO CARDHOLDER
        AGREEMENT

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008000
TSI FC00
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/CUSTOMER COPY
```

B.7.2 EMV Insert, PIN Verified

B.7.2.1 Uniterm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXX1007
auth	196123
batch	1
cardholdername	AEIPS 20 TEST CARD
cardtype	AMEX
code	AUTH
item	567
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_custom	refnum:660136000010015670
rcpt_emv_ac	5C221DC28EB72FCF
rcpt_emv_actype	TC
rcpt_emv_aid	A000000025010801
rcpt_emv_cvm	pin
rcpt_emv_name	AMERICAN EXPRESS
rcpt_emv_tsi	F800
rcpt_emv_tvr	0000008000
rcpt_entry_mode	C
rcpt_issuer_resp_code	00
rcpt_resp_code	025
timestamp	1437407332
ttid	740
u_errorcode	SUCCESS

B.7.2.2 Example Receipt

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

          SALE

MID: 1625         Lane: 1
AMEX                    C
Card:     XXXXXXXXXXX1007
Date/Time:   072015114852
Order #:            24425
TTID:                 740
Auth: 196123     Batch: 1
refnum:660136000010015670

AMOUNT:             30.00

     VERIFIED BY PIN

        APPROVED

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008000
TSI F800
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/CUSTOMER COPY
```

B.7.3 EMV Insert, No CVM

B.7.3.1 Uniterm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXX1005
auth	201507
batch	1
cardholdername	AEIPS 32/VER 1.0
cardtype	AMEX
code	AUTH
item	590
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_custom	refnum:660136000010015900
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_issuer_resp_code	00
rcpt_resp_code	025
timestamp	1437409256
ttid	764
u_errorcode	SUCCESS

B.7.3.2 Example Receipt

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

          SALE

MID: 1625         Lane: 1
AMEX                    C
Card:     XXXXXXXXXXX1005
Date/Time:   072015122056
Order #:             4753
TTID:                 764
RespCode:          00/025
Auth: 201507     Batch: 1
refnum:660136000010015900

AMOUNT:             62.00

        APPROVED

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008000
TSI F800
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/CUSTOMER COPY
```

B.7.4 EMV Insert, Card Decline

B.7.4.1 Uniterm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXX1007
batch	1
cardholdername	AEIPS 20 TEST CARD
cardtype	AMEX
code	DENY
item	584
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_custom	refnum:660136000010015850
rcpt_emv_ac	5C221DC28EB72FCF
rcpt_emv_actype	AAC
rcpt_emv_aid	A000000025010801
rcpt_emv_cvm	pin
rcpt_emv_name	AMERICAN EXPRESS
rcpt_emv_tsi	F800
rcpt_emv_tvr	0000008040
rcpt_entry_mode	C
timestamp	1437408892
ttid	758
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: 1625         Lane: 1
AMEX                    C
Card:     XXXXXXXXXXX1007
Date/Time:   072015121452
Order #:            27661
TTID:                 758
                 Batch: 1
refnum:660136000010015850

AMOUNT:             52.00

    DECLINED BY CARD
     VERIFIED BY PIN

        DECLINED

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008040
TSI F800
AAC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/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	1625
merch_name	MERCHANT NAME
merch_phone	(888) 555-1234
u_errorcode	EMV_CARD_REMOVED
verbiage	Transaction aborted - card removed

B.7.5.2 Example Receipt

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

          SALE

MID: 1625         Lane: 1
Date/Time:   072015175737
Order #:        899065992

AMOUNT:              1.00

      CARD REMOVED

        DECLINED


 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/CUSTOMER COPY
```

B.7.6 EMV Insert, Interac

B.7.6.1 Uniterm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXXX1933
auth	490708
batch	1
cardholdername	Test Card 1
cardtype	INTERAC
code	AUTH
item	623
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:660136000010016230
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	072015160749
rcpt_issuer_resp_code	00
rcpt_resp_code	001
timestamp	1437422869
ttid	804

u_errorcode	SUCCESS

B.7.6.2 Example Receipt

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

          SALE

MID: 1625         Lane: 1
INTERAC                 C
Acct Type:       CHECKING
Card:    XXXXXXXXXXXX1933
Date/Time:   072015160749
Order #:             1395
TTID:                 804
Auth: 490708     Batch: 1
refnum:660136000010016230

AMOUNT:              5.01

     VERIFIED BY PIN

        APPROVED

Interac
AID A0000002771010
TVR 8000008000
TSI 7800
TC 882D8427A268E214

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

 MERCHANT/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: XXXXXXXXXXXXXXX1311
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:    XXXXXXXXXXXX0010
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

```
/* Monetra Uniterm example program in C#
 *
 * Depends on the libmonetra C# .Net native API
 *
 * Implemented based on the Monetra 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 {
 /* Monetra Connectivity Information
  * NOTE: This is currently pointing to our Test Server that you may
  *       use for initial testing if desired.  Obviously for production,
  *       or testing with encrypted card readers, you need to point
  *       this to your local Monetra server and the username/password
  *       you configured there.  Please take note of the restrictions
  *       on the user permissions requirements in the Uniterm Guide.
  */
 private const string monetra_host     = "testbox.monetra.com";
 private const int    monetra_port     = 8665;
 private const string monetra_user     = "test_retail:public";
 private const string monetra_pass     = "publ1ct3st";


 /* 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;


 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\\Monetra Uniterm\\monetra_uniterm.exe";
   default:
    return "/usr/local/monetra/bin/monetra_uniterm";
  }
 }

 /*! Function to launch the 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) */
 static void uniterm_launch()
 {
  Process uniterm                  = new Process();
  uniterm.StartInfo.FileName       = uniterm_path();
  uniterm.StartInfo.CreateNoWindow = true;

  uniterm.Start();

  /* Make sure Uniterm is ready before returning,
   * Sleep 1000ms (1s) */
  System.Threading.Thread.Sleep(1000);
 }


 /*! 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
  */
 static Monetra uniterm_connect_host(string host, int port, ref string errorstr)
 {
  /* Initialize the Class */
  Monetra 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 the 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 (!conn.Connect()) {
   errorstr = conn.ConnectionError();
   return null;
  }

  return conn;
 }


 /*! Wrapper function to connect to Uniterm
  * \param[out] errorstr Textual error message if returns null
  * \return Initialized connection class on success. null on failure
  */
 static Monetra uniterm_connect(ref string errorstr)
 {
  Monetra conn;
  string myerror = "";
  conn = uniterm_connect_host(uniterm_host, uniterm_port, ref myerror);
  if (conn == null) {
   errorstr = "Connection to Uniterm Failed: " + myerror;
  }
  return conn;
 }


 /*! Request a transaction from Uniterm as documented in the Uniterm Guide.
  *  The Key/Value pair params are a combination of the Parameters as
  *  documented in the Uniterm Guide and the Monetra Client Interface
  *  Protocol Spec.
  * \param[in] uniterm_conn  Initialized connection to Uniterm
  *                          as returned by uniterm_connect()
  * \param[in] mparams      Array of key/value parameters to send to
  *                         Uniterm
  * \return Hashtable of string key/value pairs from response.  Please refer
  *         to the Uniterm Guide and Monetra Client Interface Protocol
  *         specification for the applicable list based on the action being
  *         performed.  "code" and "u_errorcode" are always guaranteed to
  *         be returned.
  */
 static Hashtable uniterm_sendrequest(Monetra uniterm_conn, Hashtable mparams)
 {
  int id;

  Hashtable response = new Hashtable();

  /* Request a new transaction from libmonetra */
  id = uniterm_conn.TransNew();

  /* For each item in the params hashtable, add it to the transaction */
  foreach (DictionaryEntry kv in mparams) {
   uniterm_conn.TransKeyVal(id, (String)kv.Key, (String)kv.Value);
  }

  /* Send the request to the Uniterm.  It will not return until
   * a response is available, or a disconnect is detected */
  if (!uniterm_conn.TransSend(id)) {
   /* Disconnect detected, return an appropriate error condition!
    * This should really never happen though... */
   response["code"]        = "DENY";
   response["u_errorcode"] = "CONN_ERROR";
   response["verbiage"]    = "Connection to Uniterm failed: "
                             + uniterm_conn.ConnectionError();
   return response;
  }

  /* Save the response parameters from the Uniterm into a
   * HashTable as our function prototype states. */
  string[] keys = uniterm_conn.ResponseKeys(id);
  for (int i=0; i < keys.Length; i++) {
   response[keys[i]] = uniterm_conn.ResponseParam(id, keys[i]);
  }

  /* Free up some memory by purging unneeded data */
  uniterm_conn.DeleteTrans(id);

  return response;
 }


 /*! 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 the Uniterm
  *                         as returned by uniterm_connect()
  */
 static void uniterm_shutdown(Monetra uniterm_conn)
 {
  Hashtable mparams = new Hashtable();
  mparams["u_action"] = "shutdown";
  uniterm_sendrequest(uniterm_conn, mparams);
 }


 /*! Main entry point to this application to be executed */
 static void Main()
 {
  Monetra   uniterm_conn;
  string    errorstr = "";
  Hashtable response;

  /* Step1: Launch the Uniterm */
  uniterm_launch();
  Console.WriteLine("Uniterm Launched");

  /* Step2: Connect to the Uniterm */
  uniterm_conn = uniterm_connect(ref errorstr);
  if (uniterm_conn == null) {
   Console.WriteLine("Failure: " + errorstr);
   return;
  }
  Console.WriteLine("Connected to Uniterm");


  /* Step3: Send txnrequest to Uniterm */
  Hashtable mparams = new Hashtable();
  /* 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"]     = "USB";

    /* 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";

    response = uniterm_sendrequest(uniterm_conn, mparams);
    if (String.Compare((string)response["code"], "AUTH", true) != 0) {
     Console.WriteLine("Transaction failed.");
    } else {
     Console.WriteLine("Transaction SUCCESSFUL!");
    }

    /* Print out all the response key/value pairs ... */
    foreach (DictionaryEntry kv in response) {
     Console.WriteLine("\t" + (string)kv.Key + " = " + (string)kv.Value);
    }

    /* 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 the Uniterm.
     */

    /* Step4: Cleanup */
    uniterm_shutdown(uniterm_conn);

    /* Connections will be automatically closed when the uniterm_conn
     * initialized class is closed by the destructor/garbage
     * collector */
   }

  }


```

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

```
/* Monetra Uniterm example program in C# using XML and HttpWebRequest
 *
 * Works with .Net Compact Framework v2
 *
 * Implemented based on the Monetra Uniterm Guide in conjunction with the
 * Monetra Client Interface Protocol Specification
 *
 * Please contact support@monetra.com with any questions
 */
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Threading;
using System.Collections;
using System.Net;
using System.Xml;
using System.ComponentModel;
using System.Windows.Forms;
using System.Security.Cryptography.X509Certificates;

/* NOTE: if compiling with Mono, you can use
 *        gmcs -r:System.Windows.Forms.dll utest_xml.cs
 */

class utest_xml
{
 /* Monetra Connectivity Information
  * NOTE: This is currently pointing to our Test Server that you may
  *       use for initial testing if desired.  Obviously for production,
  *       or testing with encrypted card readers, you need to point
  *       this to your local Monetra server and the username/password
  *       you configured there.  Please take note of the restrictions
  *       on the username setup as listed in section 4.2 of the
  *       Uniterm Guide (the red note at the bottom of that section).
  */
 private const string monetra_host = "testbox.monetra.com";
 private const int    monetra_port = 8665;
 private const string monetra_user = "test_retail:public";
 private const string monetra_pass = "publ1ct3st";


 /* 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;


 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\\Monetra Uniterm\\monetra_uniterm.exe";
   default:
    return "/usr/local/monetra/bin/monetra_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) */
 static void uniterm_launch()
 {
  Process monetra_uniterm = new Process();
  monetra_uniterm.StartInfo.FileName = uniterm_path();
  /* Not supported on CE
   * monetra_uniterm.StartInfo.CreateNoWindow = true;
   */

  monetra_uniterm.Start();

  /* Make sure Uniterm is ready before returning,
  * Sleep 1000ms (1s) */
  System.Threading.Thread.Sleep(1000);
 }


 /*! Trust all SSL server certificates */
 internal class AcceptAllCertificatePolicy : ICertificatePolicy
 {
  public AcceptAllCertificatePolicy()
  {
  }
  public bool CheckValidationResult(ServicePoint sPoint,
                                   X509Certificate cert,
                                   WebRequest wRequest, int certProb)
  {
   // *** Always accept
   return true;
  }
 }


 /*! Function to POST and XML message to a Monetra-like entity
 *  (Monetra or 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
 * \return True on successful communication, False if communication failed.
 *         Note: True doesn't mean the transaction itself was successful.
 */
 static Hashtable uniterm_https_post(string host, int port, string xml)
 {
  Hashtable       response = new Hashtable();
  string          url      = "https://" + host + ":" + port.ToString();
  HttpWebRequest  req      = (HttpWebRequest)WebRequest.Create(url);
  string          xmlout;

  try {
   /* POST Request */

   /* Disable SSL Server Certificate Checking */
   System.Net.ServicePointManager.CertificatePolicy =
    new AcceptAllCertificatePolicy();

   byte[] bytes;
   bytes                  = System.Text.Encoding.ASCII.GetBytes(xml);
```

```
    req.Method              = "POST";
    req.ContentType         = "text/xml";
    req.ContentLength       = bytes.Length;
    Stream reqStream        = req.GetRequestStream();
    reqStream.Write(bytes, 0, bytes.Length);
    reqStream.Close();

    /* Read Response */
    /* Note issues with .Net CF v2 as per below:
     *   http://blogs.msdn.com/b/andrewarnottms/archive/2007/11/19/why-net-compact-framework-fa
     *   http://support.microsoft.com/kb/970549
     * If the Server is OpenSSL, this can be worked around by setting
     * SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS
     */
    HttpWebResponse resp    = (HttpWebResponse)req.GetResponse();
    Stream respStream       = resp.GetResponseStream();
    StreamReader rdr        = new StreamReader(respStream);
    xmlout                  = rdr.ReadToEnd();
    rdr.Close();
   } catch (System.Net.WebException e)  {
    response["code"]         = "DENY";
    response["u_errorcode"] = "CONN_ERROR";
    response["verbiage"]     = "Connection to " + url + " failed: " +
                              e.Message;
    return response;
   }
   XmlDocument     xmldoc     = new XmlDocument();
   xmldoc.LoadXml(xmlout);

   XmlNodeList     trans      = xmldoc.DocumentElement.
                                SelectSingleNode("Resp").ChildNodes;
   foreach (XmlNode kv in trans) {
    response[kv.Name] = kv.InnerText;
   }
   return response;
  }


  /*! Request a ttransaction from Uniterm as documented in the Monetra
   *  Uniterm Guide.  The Key/Value pair params are a combination of the
   *  Parameters as documented in the Uniterm Guide and the Monetra Client
   *  Interface Protocol Spec.
   * \param[in] mparams   Array of key/value parameters to send to Uniterm
   * \return Hashtable of string key/value pairs from response.  Please refer
   *         to the Uniterm Guide and Monetra Client Interface Protocol
   *         specification for the applicable list based on the action being
   *         performed.  "code" and "u_errorcode" are always guaranteed to
   *         be returned.
   */
  static Hashtable uniterm_sendrequest(Hashtable mparams)
  {
   string XML;

   XML = "<MonetraTrans>" +
           "<Trans identifier='1'>";

   /* For each item in the params hashtable, add it to the transaction */
   foreach (DictionaryEntry kv in mparams) {
```

```
   XML = XML + "<" + (String)kv.Key + ">" + (String)kv.Value + "</" +
         (string)kv.Key + ">";
  }

  XML = XML + "</Trans></MonetraTrans>";

  return uniterm_https_post(uniterm_host, uniterm_port, XML);
 }


 /*! 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
  *  Uniterm is already running on the next execution
  *  of this application!
  */
 static void uniterm_shutdown()
 {
  Hashtable mparams = new Hashtable();
  mparams["u_action"] = "shutdown";
  uniterm_sendrequest(mparams);
 }


 /*! Main entry point to this application to be executed */
 static void Main()
 {
  Hashtable response;

  /* Step1: Launch Uniterm */
  uniterm_launch();
  MessageBox.Show("Uniterm Launched");


  /* Step2: Send txnrequest to Uniterm */
  Hashtable mparams = new Hashtable();
  /* Append the parameters for the ticket request as per the Monetra
   * Uniterm Guide, section 4 */
  mparams["username"]     = monetra_user;
  mparams["password"]     = monetra_pass;
  mparams["u_action"]     = "txnrequest";
  mparams["u_devicetype"] = "ingenico_rba";
  mparams["u_device"]     = "USB";

  /* 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";

  response = uniterm_sendrequest(mparams);
  string resultMsg = "";
  if (String.Compare((string)response["code"], "AUTH", true) != 0) {
   resultMsg = "Transaction failed.\r\n";
  } else {
   resultMsg = "Transaction SUCCESSFUL!\r\n";
  }

  /* Print out all the response key/value pairs ... */
  foreach (DictionaryEntry kv in response) {
   resultMsg = resultMsg + (string)kv.Key + " = " + (string)kv.Value +
               "\r\n";
  }

  MessageBox.Show(resultMsg);

  /* NOTE: No real reason to exit here ... we could just keep running
   *       Step 2 all day long as long.
   *       No reason to keep starting/stopping Uniterm.
   */

  /* Step3: Cleanup */
  uniterm_shutdown();

  /* Connections will be automatically closed when the uniterm_conn
   * initialized class is closed by the destructor/garbage
   * collector */
 }
}

```

C.3 Java using libmonetra

```
/* Uniterm example program in Java
 *
 * Depends on the libmonetra Java native API
 *
 * Implemented based on the Monetra Uniterm Guide in conjunction with the
 * Monetra Client Interface Protocol Specification
 *
 * Please contact support@monetra.com with any questions
 */
import java.util.Hashtable;
import java.util.Enumeration;
import com.mainstreetsoftworks.MONETRA;

/* Compile/run with:
 *   javac -classpath MONETRA.jar utest.java
 *   java -cp "./MONETRA.jar:." utest
 */

class utest {
 /* Monetra Connectivity Information
  * NOTE: This is currently pointing to our Test Server that you may
  *       use for initial testing if desired.  Obviously for production,
  *       or testing with encrypted card readers, you need to point
  *       this to your local Monetra server and the username/password
  *       you configured there.  Please take note of the restrictions
  *       on the user permission setup as described in the Monetra Uniterm
  *       Guide.
  */
 private static String monetra_host     = "testbox.monetra.com";
 private static int    monetra_port     = 8665;
 private static String monetra_user     = "test_retail:public";
 private static String monetra_pass     = "publ1ct3st";


 /* 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 static String uniterm_host         = "localhost";
 private static int    uniterm_port         = 8123;

 static String uniterm_path()
 {
  if (System.getProperty("os.name").startsWith("Windows")) {
   return "C:\\Program Files\\Main Street Softworks\\Monetra Uniterm\\monetra_uniterm.exe";
  } else {
   return "/usr/local/monetra/bin/monetra_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) */
 static void uniterm_launch()
 {
  try {
   Process p = new ProcessBuilder(uniterm_path()).start();
  } catch (java.io.IOException e) {
   System.out.println(e.getMessage());
   System.exit(1);
  }
  /* Make sure Uniterm is ready before returning,
   * Sleep 1000ms (1s) */
  try {
   Thread.sleep(1000);
  } catch (InterruptedException e) {
  }
 }


 /*! 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
  */
 static MONETRA uniterm_connect_host(String host, int port,
                                     StringBuilder errorstr)
 {
  /* Initialize the Class */
  MONETRA conn = new MONETRA("");

  errorstr.setLength(0);

  /* 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(0);

  /* 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(1);

  /* Connect! */
  if (conn.Connect() == 0) {
   errorstr.append(conn.ConnectionError());
   return null;
  }

  return conn;
 }


 /*! Wrapper function to connect to Uniterm
  * \param[out] errorstr Textual error message if returns null
  * \return Initialized connection class on success. null on failure
  */
 static MONETRA uniterm_connect(StringBuilder errorstr)
 {
  MONETRA conn;
  StringBuilder myerror = new StringBuilder();
  conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror);
  if (conn == null) {
   errorstr.setLength(0);
   errorstr.append("Connection to Uniterm Failed: " +
                   myerror.toString());
  }
  return conn;
 }


 /*! Request a transaction from Uniterm as documented in the Monetra
  *  Uniterm Guide.  The Key/Value pair params are a combination of the
  *  Parameters as documented in the Uniterm Guide and the Monetra Client
  *  Interface Protocol Spec.
  * \param[in] uniterm_conn  Initialized connection to Uniterm
  *                          as returned by uniterm_connect()
  * \param[in] mparams       Array of key/value parameters to send to
  *                          Uniterm
  * \return Hashtable of string key/value pairs from response.  Please refer
  *         to the Uniterm Guide and Monetra Client Interface Protocol
  *         specification for the applicable list based on the action being
  *         performed.  "code" and "u_errorcode" are always guaranteed to
  *         be returned.
  */
 static Hashtable<String,String> uniterm_sendrequest(MONETRA uniterm_conn,
        Hashtable<String,String> mparams)
 {
  long id;

  Hashtable response = new Hashtable<String,String>();

  /* Request a new transaction from libmonetra */
  id = uniterm_conn.TransNew();

  /* For each item in the params hashtable, add it to the transaction */
  for (String key : mparams.keySet()) {
   String value = mparams.get(key);
   uniterm_conn.TransKeyVal(id, key, value);
  }

  /* Send the request to the Uniterm.  It will not return until
   * a response is available, or a disconnect is detected */
  if (uniterm_conn.TransSend(id) == 0) {
   /* Disconnect detected, return an appropriate error condition!
    * This should really never happen though... */
   response.put("code",         "DENY");
   response.put("u_errorcode", "CONN_ERROR");
   response.put("verbiage",     "Connection to Uniterm failed:"
                               + uniterm_conn.ConnectionError());
   return response;
  }

  /* Save the response parameters from the Uniterm into a
   * HashTable as our function prototype states. */
  String[] keys = uniterm_conn.ResponseKeys(id);
  for (int i=0; i < keys.length; i++) {
   response.put(keys[i], uniterm_conn.ResponseParam(id, keys[i]));
  }

  /* Free up some memory by purging unneeded data */
  uniterm_conn.DeleteTrans(id);

  return response;
 }


 /*! 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()
  */
 static void uniterm_shutdown(MONETRA uniterm_conn)
 {
  Hashtable mparams = new Hashtable<String,String>();
  mparams.put("u_action", "shutdown");
  uniterm_sendrequest(uniterm_conn, mparams);
 }

```

```

/*! Main entry point to this application to be executed */
public static void main(String[] args)
{
 MONETRA                  uniterm_conn;
 StringBuilder            errorstr = new StringBuilder();
 Hashtable<String,String> response;
 String                   ticket;

 /* Step1: Launch Uniterm */
 uniterm_launch();
 System.out.println("Uniterm Launched");

 /* Step2: Connect to Uniterm */
 uniterm_conn = uniterm_connect(errorstr);
 if (uniterm_conn == null) {
  System.out.println("Failure: " + errorstr.toString());
  return;
 }
 System.out.println("Connected to Uniterm");

 /* Step3: Send a txnrequest to Uniterm */
 Hashtable<String,String> mparams = new Hashtable<String,String>();
 /* Append the parameters for the txnrequest */
 mparams.put("username",     monetra_user);
 mparams.put("password",     monetra_pass);
 mparams.put("u_action",     "txnrequest");

 mparams.put("u_devicetype", "ingenico_rba");
 mparams.put("u_device",     "USB");

 /* 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.put("action",    "sale");
 mparams.put("amount",    "12.00");
 mparams.put("ordernum", "123456");
 mparams.put("comments", "u_txnrequest");

 response = uniterm_sendrequest(uniterm_conn, mparams);
 if (!response.get("code").equalsIgnoreCase("AUTH")) {
  System.out.println("Transasction failed.");
 } else {
  System.out.println("Transasction SUCCESSFUL!");
 }

 /* Print out all the response key/value pairs ... */
 for (String key : response.keySet()) {
  String value = response.get(key);
  System.out.println("\t" + key + " = " + value);
 }

 /* 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 the Uniterm.
  */

 /* Step4: Cleanup */
 uniterm_shutdown(uniterm_conn);

 /* Connections will be automatically closed when the uniterm_conn
  * initialized classe is closed by the destructor/garbage
  * collector */
}

}


```

C.4 PHP using libmonetra

```
<?php
/* Monetra Uniterm example program in PHP
 *
 * Depends on the libmonetra PHP native API
 *
 * Implemented based on the Monetra Uniterm Guide in conjunction with the
 * Monetra Client Interface Protocol Specification
 *
 * Please contact support@monetra.com with any questions
 */
error_reporting(E_ALL);
require_once("libmonetra.php");

/* Monetra Connectivity Information
 * NOTE: This is currently pointing to our Test Server that you may
 *       use for initial testing if desired.  Obviously for production,
 *       or testing with encrypted card readers, you need to point
 *       this to your local Monetra server and the username/password
 *       you configured there.  Please take note of the restrictions
 *       on the user permissions as documented in the Uniterm Guide.
 */
$monetra_host     = "testbox.monetra.com";
$monetra_port     = 8665;
$monetra_user     = "test_retail:public";
$monetra_pass     = "publ1ct3st";

/* 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;

/* 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\\Monetra Uniterm\\monetra_u
} else {
 /* Unix path */
 $uniterm_path         = "/usr/local/monetra/bin/monetra_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
 *                      connection to
 * \param[out] errorstr Textual error message if returns null
 * \return Initialized connection on success. null on failure
 */
function uniterm_connect_host($host, $port, &$errorstr)
{
 /* Initialize the Connection */
 $conn = M_InitConn();

 $errorstr = "";

 /* We always want to use an SSL connection to Monetra and Uniterm */
 M_SetSSL($conn, $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 */
 M_VerifySSLCert($conn, 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 */
 M_SetBlocking($conn, true);

 /* Connect! */
 if (!M_Connect($conn)) {
  $errorstr = M_ConnectionError($conn);
  return null;
 }

 return $conn;
}


/*! Wrapper function to connect to Uniterm
 * \param[out] errorstr Textual error message if returns null
 * \return Initialized connection on success. null on failure
 */
function uniterm_connect(&$errorstr)
{
 global $uniterm_host, $uniterm_port;

 $myerror = "";
 $conn = uniterm_connect_host($uniterm_host, $uniterm_port, &$myerror);
 if ($conn == null) {
  $errorstr = "Connection to Uniterm Failed: " . $myerror;
 }
 return $conn;
}


/*! Request a transaction from Uniterm as documented in the Uniterm Guide.
 *  The Key/Value pair params are a combination of the Parameters as
 *  documented in the Uniterm Guide and the Monetra Client Interface Protocol
 *  Spec.
 * \param[in] uniterm_conn  Initialized connection to Uniterm as returned by
 *                          uniterm_connect()
 * \param[in] params        Array of key/value parameters to send to Uniterm
 *
 * \return Array of string key/value pairs from response.  Please refer to the
 *         Uniterm Guide and Monetra Client Interface Protocol specification
 *         for the applicable list based on the action being performed.
 *         "code" and "u_errorcode" are always guaranteed to be returned.
 */
function uniterm_sendrequest($uniterm_conn, $params)
{
 $response = array();

 /* Request a new transaction from libmonetra */
 $id = M_TransNew($uniterm_conn);

 /* For each item in the params array, add it to the transaction */
 foreach ($params as $key => $value) {
  M_TransKeyVal($uniterm_conn, $id, $key, $value);
 }

 /* Send the request to the Uniterm.  It will not return until a
  * response is available, or a disconnect is detected */
 if (!M_TransSend($uniterm_conn, $id)) {
  /* Disconnect detected, return an appropriate error condition!
   * This should really never happen though... */
  $response["code"]        = "DENY";
  $response["u_errorcode"] = "CONN_ERROR";
  $response["verbiage"]    = "Connection to Uniterm failed: " .
                             M_ConnectionError($uniterm_conn);
  return $response;
 }

 /* Save the response parameters from the Uniterm into a HashTable
  * as our function prototype states. */
 $keys = M_ResponseKeys($uniterm_conn, $id);
 foreach ($keys as $value) {
  $response[$value]      = M_ResponseParam($uniterm_conn, $id, $value);
 }

 /* Free up some memory by purging unneeded data */
 M_DeleteTrans($uniterm_conn, $id);

 return $response;
}


/*! 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()
 */
function uniterm_shutdown($uniterm_conn)
{
 uniterm_sendrequest($uniterm_conn, array("u_action" => "shutdown"));
}



/* CODE TO EXECUTE ... */

$errorstr = "";

/* Step1: Launch Uniterm */
uniterm_launch();
echo "Uniterm Launched\r\n";

/* Step2: Connect to Uniterm */
$uniterm_conn = uniterm_connect(&$errorstr);
if ($uniterm_conn == null) {
 echo "Failure: " . $errorstr . "\r\n";
 return;
}

echo "Connected to Uniterm\r\n";


/* Step3: Send a txnrequest to the Uniterm */
$params  = array();

/* Append the parameters for the txnrequest */
$params["username"]     = $monetra_user;
$params["password"]     = $monetra_pass;
$params["u_action"]     = "txnrequest";
$params["u_devicetype"] = "ingenico_rba";
```

```
$params["u_device"]     = "USB";

/* 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 */
$params['action']   = 'sale';
$params['amount']   = '12.00';
$params['ordernum'] = '123456';
$params['comments'] = 'u_txnrequest';

$response = uniterm_sendrequest($uniterm_conn, $params);
if (strcasecmp($response["code"], "AUTH") != 0) {
 echo "Transaction Failed.\r\n";
} else {
 echo "Transaction SUCCESSFUL!\r\n";
}

/* Print out all the response key/value pairs ... */
foreach ($response as $key => $value) {
 echo "\t" . $key . " = " . $value . "\r\n";
}

/* 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 connection is closed by the destructor/garbage collector */

?>


```

C.5 Microsoft VB.Net using libmonetra

```
' Monetra Uniterm example program in VB.Net
'
' Depends on the libmonetra C# .Net native API (DLL)
'
' Implemented based on the Monetra Uniterm Guide in conjunction with the
' Monetra Client Interface Protocol Specification
'
' Please contact support@monetra.com with any questions

Option Explicit On
Option Strict On

Imports System
Imports System.Collections
Imports System.Diagnostics
Imports System.Threading
Imports libmonetra

' On unix, compile using:
'   gmcs /target:library /unsafe libmonetra.cs
'   vbnc2 -r:libmonetra.dll utest.vb

Module Module1
 ' Monetra Connectivity Information
 ' NOTE: This is currently pointing to our Test Server that you may
 '       use for initial testing if desired.  Obviously for production,
 '       or testing with encrypted card readers, you need to point
 '       this to your local Monetra server and the username/password
 '       you configured there.  Please take note of the restrictions
 '       on the user permissions as documented in the Uniterm Guide.
 Private Const monetra_host As String = "testbox.monetra.com"
 Private Const monetra_port As Integer = 8665
 Private Const monetra_user As String = "test_retail:public"
 Private Const monetra_pass As String = "publ1ct3st"


 ' 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 uniterm_host As String = "localhost"
 Private Const uniterm_port As Integer = 8123


 Private Function uniterm_path As String
  Select Case Environment.OSVersion.Platform
   Case PlatformID.Win32NT, PlatformID.Win32S, _
        PlatformID.Win32Windows, PlatformID.WinCE
    Return "C:\\Program Files\\Main Street Softworks\\Monetra Uniterm\\monetra_uniterm.exe"
   Case Else
    Return "/usr/local/monetra/bin/monetra_uniterm"
  End Select
 End Function


 '! 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)
 Private Sub uniterm_launch()
  Dim monetra_uniterm As New Process()
  monetra_uniterm.StartInfo.FileName = uniterm_path
  monetra_uniterm.StartInfo.CreateNoWindow = True

  monetra_uniterm.Start()

  ' Make sure Uniterm is ready before returning,
  ' Sleep 1000ms (1s)
  System.Threading.Thread.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
 Private Function uniterm_connect_host(ByVal host As String, ByVal port _
                                   As Integer, ByRef errorstr As String) _
                                   As Monetra
  ' Initialize the Class
  Dim conn As 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()
   Return Nothing
  End If

  Return conn
 End Function


 '! Wrapper function to connect to the Uniterm
 ' \param[out] errorstr Textual error message if returns null
 ' \return Initialized connection class on success. null on failure
 Private Function uniterm_connect(ByRef errorstr As String) As Monetra
  Dim conn As Monetra
  Dim myerror As String = ""
  conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror)
  If conn Is Nothing Then
   errorstr = "Connection to Uniterm Failed: " + myerror
  End If

  Return conn
 End Function

 ' Request a transaction from Uniterm as documented in the Uniterm Guide.
 ' The Key/Value pair params are a combination of the Parameters as
 ' documented in the Uniterm Guide and the Monetra Client Interface
 ' Protocol Spec.
 ' \param[in] uniterm_conn  Initialized connection to the Uniterm
 '                          as returned by uniterm_connect()
 ' \param[in] mparams      Array of key/value parameters to send to Uniterm
 ' \return Hashtable of string key/value pairs from response.  Please refer
 '         to the Uniterm Guide and Monetra Client Interface Protocol
 '         specification for the applicable list based on the action being
 '         performed.  "code" and "u_errorcode" are always guaranteed to
 '         be returned.
 Private Function uniterm_sendrequest(ByVal uniterm_conn As Monetra, ByVal _
                                      mparams As Hashtable) As Hashtable
  Dim id As Integer
  Dim response As New Hashtable

  ' Request a new transaction from libmonetra
  id = uniterm_conn.TransNew()

  ' For each item in the params hashtable, add it to the transaction
  Dim kv As DictionaryEntry
  For Each kv In mparams
   uniterm_conn.TransKeyVal(id, CType(kv.Key, String), _
                            CType(kv.Value, String))
  Next kv

  ' Send the request to the Uniterm.  It will not return until a
  ' response is available, or a disconnect is detected
  If Not uniterm_conn.TransSend(id) Then
   ' Disconnect detected, return an appropriate error condition!
   ' This should really never happen though...
   response("code")        = "DENY"
   response("u_errorcode") = "CONN_ERROR"
   response("verbiage")    = "Connection to Uniterm failed:" _
                             + uniterm_conn.ConnectionError()
   Return response
  End If

  ' Save the response parameters from Uniterm into a
  ' HashTable as our function prototype states. */
  Dim keys() As String = uniterm_conn.ResponseKeys(id)
  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")    = "USB"

  ' 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"

  response = uniterm_sendrequest(uniterm_conn, mparams)
  If StrComp(CType(response("code"), String), "AUTH", _
                   vbTextCompare) <> 0 Then
   Console.WriteLine("Transaction failed.")
  Else
   Console.WriteLine("Transaction SUCCESSFUL!")
  End If

  ' Print out all the response key/value pairs ...
  Dim kv As DictionaryEntry
  For Each kv In response
   Console.WriteLine("  " + CType(kv.Key, String) + " = " + _
                     CType(kv.Value, String))
  Next kv

  ' 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 closed by the destructor/garbage
  ' collector
 End Sub

End Module


```

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

```
' Monetra Uniterm example program in VBScript
'
' Depends on the MSXML, and Microsoft Scripting Runtime
'
' Implemented based on the Monetra Uniterm Guide in conjunction with the
' Monetra Client Interface Protocol Specification
'
' Please contact support@monetra.com with any questions

Option Explicit

' Monetra Connectivity Information
Dim monetra_host
Dim monetra_port
Dim monetra_user
Dim monetra_pass

' Uniterm Connectivity Information
Dim uniterm_host
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)

 Dim Trans
 Set Trans = xmldoc.documentElement.selectSingleNode("Resp").childNodes

 Dim kv
 For Each kv In Trans
  myresponse(kv.nodeName) = kv.text
 Next

 uniterm_https_post = True
End Function


'! Request a transaction from Uniterm as documented in the Uniterm Guide.
'  The Key/Value pair params are a combination of the Parameters as documented
'  the Uniterm Guide and the Monetra Client Interface Protocol Spec.
' \param[in]  mparams    Dictionary of key/value parameters to send to the
'                        Uniterm
' \param[out] errorstr   If returning False, the error message, typically comms
'                         error
' \param[out] myresponse Dictionary of string key/value pairs from response.
'                         Please refer to the Uniterm Guide and Monetra Client
'                         Interface Protocol specification for the applicable
'                         list based on the action being performed.  "code" and
'                         "u_errorcode" are always guaranteed to be returned.
' \return True on successful communication, False if communication failed.
'          Note: True doesn't mean the transaction itself was successful.
Function uniterm_sendrequest(ByVal mparams, ByRef errorstr, ByRef myresponse)
 Dim xml

 xml = "<MonetraTrans><Trans identifier='1'>"

 ' For each item in the params dictionary, add it to the transaction
 Dim key
 For Each key In mparams
  xml = xml & "<" & key & ">" & mparams(key) & "</" & key & ">"
 Next

 xml = xml & "</Trans></MonetraTrans>"

 uniterm_sendrequest = uniterm_https_post(uniterm_host, uniterm_port, xml, _
                                          errorstr, myresponse)
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!
Sub uniterm_shutdown()
 Dim myresponse
 Dim errorstr
 Dim mparams

 Set mparams = CreateObject("Scripting.Dictionary")
 mparams("u_action") = "shutdown"

 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
' NOTE: This is currently pointing to our Test Server that you may
'       use for initial testing if desired.  Obviously for production,
'       or testing with encrypted card readers, you need to point
'       this to your local Monetra server and the username/password
'       you configured there.  Please take note of the restrictions
'       on the user permissions as documented in the Uniterm Guide.
monetra_host = "testbox.monetra.com"
monetra_port = 8665
monetra_user = "test_retail:public"
monetra_pass = "publ1ct3st"

' 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\\Monetra Uniterm\\monetra_uniterm.exe

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")     = "USB"

' 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

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

MsgBox (msg)


' NOTE: No real reason to exit here ... we could just keep running
'       Step 2 all day long.  No reason to keep starting/stopping the
'       Uniterm.

' Step3: Cleanup
uniterm_shutdown





```

C.7 Microsoft Visual Basic 6 using libmonetra

```
Attribute VB_Name = "Module1"
' Monetra Uniterm example program in VB6
'
' Depends on the libmonetra C# .Net native API (DLL) (has COM hooks)
'
' Must add reference to libmonetra and Microsoft Scripting Runtime
'
' Implemented based on the Monetra Uniterm Guide in conjunction with the
' Monetra Client Interface Protocol Specification
'
' Please contact support@monetra.com with any questions

Option Explicit

' MonetraInformation
Dim monetra_user As String
Dim monetra_pass As String

' Uniterm Connectivity Information
Dim uniterm_host As String
Dim uniterm_port As Integer
Dim uniterm_path As String

Private Declare Sub Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)

'! 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
 Dim conn As IMonetra
 Dim myerror As String

 myerror = ""
 Set conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror)
 If conn Is Nothing Then
  errorstr = "Connection to Uniterm Failed: " & myerror
 End If
 Set uniterm_connect = conn
End Function


' Request a transaction from Uniterm as documented in the Monetra Uniterm
' Guide.  The Key/Value pair params are a combination of the Parameters as
' Uniterm Guide and the Monetra Client Interface Protocol Spec.
' \param[in] uniterm_conn  Initialized connection to Unitermas returned by
'                          connect_to_uniterm()
' \param[in] mparams       Dictionary of key/value parameters to send to
'                          Uniterm
' \return Dictionary of string key/value pairs from response.  Please refer
'  to the Uniterm Guide and Monetra Client Interface Protocol
'  specification for the applicable list based on the action being
'  performed.  "code" and "u_errorcode" are always guaranteed to
'  be returned.
Function uniterm_sendrequest(ByVal uniterm_conn As IMonetra, _
    ByVal mparams As Dictionary) _
    As Dictionary
 Dim id As Integer
 Dim myresponse As New Dictionary

 ' Request a new transaction from libmonetra
 id = uniterm_conn.TransNew()

 ' For each item in the params dictionary, add it to the transaction
 Dim key
 For Each key In mparams
  uniterm_conn.TransKeyVal id, key, mparams(key)
 Next key

 ' Send the request to the Uniterm.  It will not return until a
 ' response is available, or a disconnect is detected
 If Not uniterm_conn.TransSend(id) Then
  ' Disconnect detected, return an appropriate error condition!
  ' This should really never happen though...
  myresponse("code") = "DENY"
  myresponse("u_errorcode") = "CONN_ERROR"
  myresponse("verbiage") = "Connection to Uniterm failed: " _
      & uniterm_conn.ConnectionError()
  Set uniterm_sendrequest = myresponse
  Exit Function
 End If

 ' Save the response parameters from the Uniterm into a HashTable
 ' as our function prototype states.
 Dim keys() As String
 keys = uniterm_conn.ResponseKeys(id)
 Dim i As Integer
 For i = LBound(keys) To UBound(keys)
  myresponse(keys(i)) = uniterm_conn.ResponseParam(id, keys(i))
 Next i

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

 Set uniterm_sendrequest = myresponse
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 Uniterm is already running on the next execution
'  of this application!
' \param[in] uniterm_conn Initialized connection to Uniterm as returned by
'             connect_to_uniterm()
Sub uniterm_shutdown(ByVal uniterm_conn As IMonetra)
```

```
 Dim mparams As New Dictionary

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


'! Main entry point to this application to be executed
Sub Main()
 ' NOTE: This is currently pointing to our Test Server that you may
 '       use for initial testing if desired.  Obviously for production,
 '       or testing with encrypted card readers, you need to point
 '       this to your local Monetra server and the username/password
 '       you configured there.  Please take note of the restrictions
 '       on the user permissions as documented in the Uniterm Guide.
 monetra_host = "testbox.monetra.com"
 monetra_port = 8665
 monetra_user = "test_retail:public"
 monetra_pass = "publ1ct3st"

 ' 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\\Monetra Uniterm\\monetra_uniterm.ex

 Dim uniterm_conn As IMonetra
 Dim errorstr As String
 Dim myresp As Dictionary
 Dim msg As String

 errorstr = ""

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

 ' Step2: Connect to Uniterm
 Set uniterm_conn = uniterm_connect(errorstr)
 If uniterm_conn Is Nothing Then
  MsgBox ("Failure: " & errorstr)
  Exit Sub
 End If

 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")     = "USB"

 ' 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.

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	UniTerm does not provide or facilitate administrative access.

administrative access and access to cardholder data.	
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.
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 SSL/TLS using proprietary protocols.
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 SSL/TLS using proprietary protocols.
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.