UniTerm® Payment Interface Application

UniTerm Integration and Deployment Guide

Revision: 9.4.1

Publication date August 20, 2019

UniTerm Integration and Deployment Guide

Monetra Technologies, LLC

Revision: 9.4.1

Publication date August 20, 2019
Copyright © 2019 Monetra Technologies, LLC

Legal Notice

The information contained herein is provided *As Is* without warranty of any kind, express or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. There is no warranty that the information or the use thereof does not infringe a patent, trademark, copyright, or trade secret.

Monetra Technologies, LLC. SHALL NOT BE LIABLE FOR ANY DIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, WHETHER RESULTING FROM BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, OR OTHERWISE, EVEN IF MONETRA TECHNOLOGIES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. MONETRA TECHNOLOGIES RESERVES THE RIGHT TO MAKE CHANGES TO THE INFORMATION CONTAINED HEREIN AT ANYTIME WITHOUT NOTICE. NO PART OF THIS DOCUMENT MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, FOR ANY PURPOSE, WITHOUT THE EXPRESS WRITTEN PERMISSION OF Monetra Technologies, LLC.

Table of Contents

---

1 Revision History

Version	Date	Changes
v9.4.1	2019-08-20	• Better document Android HID support
v9.4.0	2019-07-08	• A new UniTerm action of `txnquick` has been added which acts like `txnrequest`, but uses the device flow for Quick Chip like `txnstart + txnfinish` to allow early removal of the card. • Support "standalone" mode of operation for Ingenico devices where the device is not API-controllable but instead acts as a standalone terminal that can only be interacted with via the device menus. Terminal must connect using `IPClient` mode. • `TCPX` firmware support on Ingenico terminals for Canada. • Tsys `RBA` recertification. • Moneris `TCPX` certification. • `RBA` recommended firmware version updates. • `UPP` recommended firmware version updates.
v9.3.0	2019-04-15	• Wording changes as requested by PCI PA-DSS auditors for clarity • Heartland Ingenico `RBA` certification listing • Update recommended `RBA` version • Dynamically generated approval code support for Stand-In requests • Receipt types can now take a pipe separated list of types to allow multiple receipt formats to be returned simultaneously • Replace `[monetra]` with `[payment_server]` in configuration section. Fallback exists for older `ini` files.
v9.2.0	2019-01-18	• Receipt updates for Moneris • Multiple TLS certificates can be loaded to support different hostnames (`SNI`) • New `u_action` of `expirecached` • US Debit rules relaxed for Stand-In processing
v9.1.0	2018-12-13	• Company Legal name changed from Main Street Softworks Inc to Monetra Technologies LLC • Removal of `ENCRYPTEDONLY u_flags`, functionality moved to Monetra merchant account setup. • JSON over HTTPS is now an available communication mechanism. • Note that `PNG` signatures can now be requested from Monetra • Add back `u_cardclass` as a response parameter as it is used to differentiate `EBTCB` vs `EBTFS` which are sub classes of the `EBT` card type. • Best practices guidance for Parking systems Card-In/Card-Out Entry and Exit gates • New `reqinput` entry type of `tip`. • New `CARDMETA u_flags` option for Global BIN checking.

Version	Date	Changes
		• Pre-formatted receipts can now be returned in responses as plain text (with user-specified width), HTML, XML, and JSON • Update `RBA` recommended versions • List IDTech and Ingenico `UPP` devices, as well as new certifications
v9.0.2	2018-05-29	• Removal of some references that no longer applies to UniTerm v9. • Further clarify stand-in configuration.
v9.0.0	2018-04-10	• Note that UniTerm v9 requires access to Monetra v8.7 or higher to operate. • New EMV certifications listed. • Apple iOS MFi (Made for iOS) no longer supported as if it was a BlueTooth device, new MFi device type. • Addition of Bluetooth LE communication method. • Removed `uniterm.ini` configuration items which are instead configured Monetra server-side now: • `tippercent` • `cashbackamount` • `cashbackmax` • New `uniterm.ini` configuration settings (SSL/TLS, shared secret, database, logging, tab, stand-in, blacklist). • New feature of Stand-In/Store and Forward processing added. • New feature of ChipTab/Bar Tab processing added. • Document new storage/database, key management, and logging facilities. • Shared Secret usage has changed to using `HMAC-SHA256`, added new section for clarity. • Removal of `u_cardclass` as it is not possible to do with EMV as the terminal will auto-select the best `AID`. POS vendors should not have a "tender type" button in their POS but instead rely on the response from UniTerm. • Update of PCI PA-DSS Security cross-reference appendix. • List BBPos/Anywhere Commerce device
v8.4.0	2017-08-21	• `signaturetimeout` configuration parameter for determining the action to take when a user forgets to sign. • New EMV Kernel for Ingenico `RBA` with new certifications • Updates to the `unsupportedcard` configuration value with additional options. • Update `u_status` codes to be more descriptive, this removes `CARD` as a response parameter in order to add further clarification on card presentation being requested. • Add `u_represent` reason field for status requests to indicate a reason for why card re-presentation is being requested. • Add `cardwait` configuration value for `[uniterm]` to configure timeout duration for the card prompt (mostly for unattended environments).

Version	Date	Changes
		• Add new receipt response parameter `rcpt_emv_pinbypass` to indicate the cardholder explicitly requested to bypass pin entry. • New Device Server mode, where the Ethernet-connected device connects outbound to UniTerm (instead of UniTerm initiating the connection to the device). Connected devices are addressed by serial number. Supports both unencrypted IP and SSL/TLS. • `u_action=reqinput` now allows requesting a zip code • Add pole display support for use with Quick Chip (`txnstart`) transactions.
v8.3.2	2017-04-14	• iOS UniTerm is now available on the iOS App Store. • Recommended Ingenico `RBA` version is now 19.0.8.
v8.3.0	2017-03-06	• New First Data certification for EMV Debit and EMV Contactless • Update recommended `RBA` versions. • Add new `cdcvm` card holder verification method response.
v8.2.5	2017-01-19	• Additional clarifications for use of the iOS framework. • Add `DELAYRESPONSE` `u_flags`. • Add `fullscreen` modifier to `guimode` configuration.
v8.2.4	2016-12-21	• Clarifications for use of the iOS framework. • Various other clarifications and fixes for typos. • Addition of `u_flowflags` response parameter.
v8.2.3	2016-12-01	• Add `serialquirks` flags for `uniterm.ini`
v8.2.2	2016-11-15	• Add `u_foodamount` value of `maybe` for `txnstart`. • `u_cardclass=EBT` has been split into `EBTFS` and `EBTCB` for clarity. • Add `[uniterm]` configuration parameter `returnbin` to return the first 6 digits (in addition to the last 4 digits) of a card number in the `txnstart` response message so transaction decisions can be made (e.g. health benefits qualification).
v8.2.0	2016-10-17	• All references to monetra_uniterm have been changed to just uniterm, including executable names and paths. • Added EMV industry question • Split out `u_action` values and descriptions into their own section for clarity. • GUI can now use Device Licensing when passing a `u_flags` or `GUIONLY` while also passing valid values for `device` and `devicetype`. This prevents requiring duplicate licensing for workstations with physical devices that sometimes perform gui card entry. • Ingenico `RBA` recommended version is now 18.04 • Linux HID device permission section has been added • Asynchronous processing (multiplexing/interleaving transactions on a single connection) is now supported. • QuickChip overview.

Version	Date	Changes
		• USB is now known as HID, all usb references changed, `hidlist` report parameters and `u_device` format have also changed • New commands (`u_action` values): • `txnstart` • `txnfinish` • `deviceupload` • `deviceprint` • `devicereboot` • `reqsignature` • `reqconfirm` • `reqinput` Associated request parameters for these new commands: • `u_b64data` • `u_filename` • `u_text` • `u_message` • `u_inputtype` Associated response parameters for these new commands: • `u_needreboot` • `u_confirmed` • `u_input` • `u_signature` • Interchange requirement information. • New/Changed configuration parameters: • `[monetra] persist_conn` - Default changed to "no" • `[uniterm] tippercent` • `[uniterm] cashbackamount` • `[uniterm] cashbackmax` • Note that `uniterm.ini` isn't actually distributed anymore, instead a template of `uniterm_example.ini` is provided. • New `u_flags`: • `NOTIP` • `NOCASHBACK` • `NOCONFIRM` • `NOSIGNATURE` • Section added on Tip processing, including the addition of the `u_tip` response parameter. • Section added on Cash Back processing, including the addition of the `u_cashback` response parameter. • Section added on EBT processing, including the addition of the `u_foodamount` request parameter and `u_wasfood` response parameter. • The OS-provided SSL/TLS root trust list is now automatically loaded and there is no longer an included `caroots` directory. • In Deployment, discuss the new standalone installers provided.

Version	Date	Changes
		• Update `devicetypes` functionality values. • Update Apple iOS integration information for the framework variant. • New `Ingenico RBA` form of `UTASEL.K3Z` used. • Add guidance for Pay at the Table. • Add information about automatic signature capture. • Note that `CPX/uCPX` is limited to UniTerm v8.0, and not supported on v8.2 • Configuration parameter `nosigfloor` has been changed to `nocvmfloor` • `modifyconfig` parameters now use "`.`" as a delimiter instead of "`/`" • `u_action=status` request now returns a `u_status` machine-readable status code • Add section on updating `RBA` firmware. • Add `skipped rcpt_emv_cvm` return value
v8.0.3	2016-03-02	• Note limitation for asynchronous processing (or lack thereof) • Update status of various certifications • Add sections relating to iOS support • Updated device application version support
v8.0.2	2015-11-16	• Update status of various certifications • Add section on obtaining devices. • `UID_NOT_FOUND` error code addition, and clarify possible `u_errorcode` values for status requests. • Missing required permission of `CARDTYPE` added. • Added `persist_conn` configuration parameter for the `[monetra]` section. • Added `deviceinfo` transaction type to request information about the connected device. • Added `GIFTPIN u_flags` to prompt for PIN entry if a gift card is used.
v8.0.1	2015-09-28	• Re-word and clarify configuration information for `Ingenico RBA` • Clarify fields can be sent in to pre-populate key entry fields in GUI mode. • Receipt should show "CALL ISSUER" for a response code of "CALL". • Cardholder Name on receipts can be printed under the signature line. • Document how to configure an `RBA` device for contactless. • Remove unused `monetra_host` and `monetra_port` variables in example code. • Add touchscreen mode support via `[uniterm]` config option of `guimode=touchscreen`. • SSL is no longer a valid communication option for Android. Only Service communication is allowed.

Version	Date	Changes
		• Add `ssl_auth_key` and `ssl_auth_cert` configuration parameters for the `[monetra]` section. • Add `ssl_cert_validate` configuration parameter for the `[monetra]` section. • Add `ssl_cadir` configuration parameter for the `[monetra]` section. • `Verifone` recommended `XPI` version updated to 8.24D. • `Ingenico` recommended `RBA` version updated to 15.06. • `Ingenico RBA` added support for `USB-HID`. • Update receipt examples to the latest generated by UniTerm Tester.
v8.0.0	2015-08-17	• Initial revision

2 UniTerm System

2.1 Overview

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

2.2 UniTerm Architecture

The UniTerm module is accessed via its 'Transaction Request' mode, as described below:
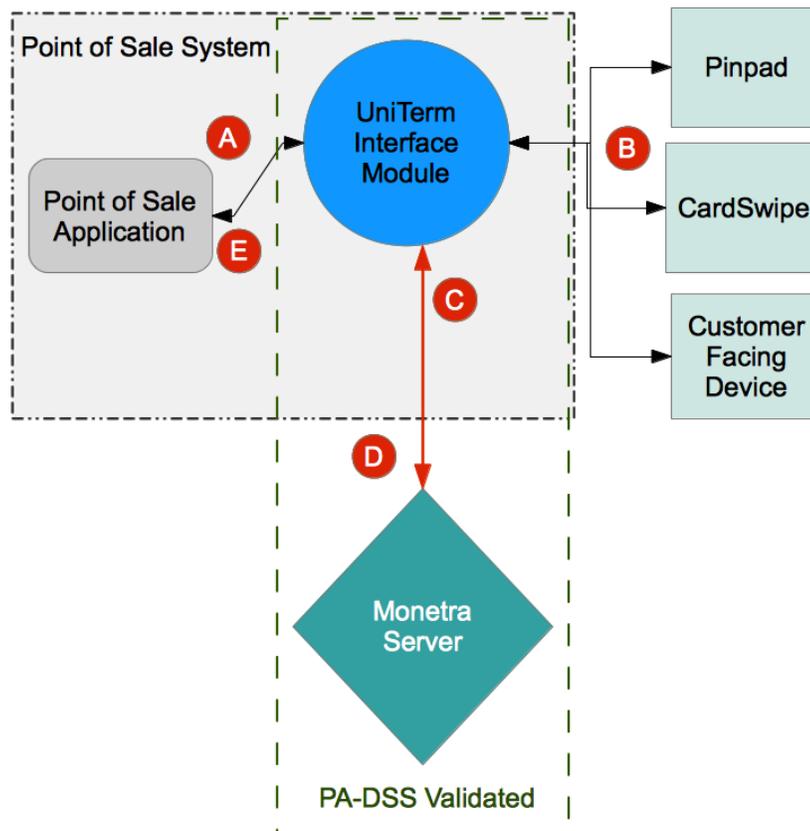
A	Point of sale application calls UniTerm for `txnrequest` (such as a sale transaction request) and includes basic information such as the amount of the

	sale and an order-number. This communication channel or request will NEVER contain sensitive cardholder data meaning it is out of PCI scope.
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 Payment Server for further processing.
D	The Payment 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. This communication channel or request will NEVER contain sensitive cardholder data meaning it is out of PCI scope.

2.3 Design Decisions

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

Note: Assuming a POS exclusively uses UniTerm to retrieve and process card holder data, and is guaranteed to not have the capability to retrieve or store card holder data, including keyed entry in the event of phone orders or similar (which should be requested through UniTerm instead), then the POS would be considered Out of Scope for PCI PA-DSS as UniTerm will never relay any sensitive card holder data to the POS.

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

2.4 Store and Forward/Stand-In processing

Store and Forward or Stand-In processing is a mechanism to locally approve a transaction under a configurable dollar amount if an unrecoverable connectivity failure to your processing institution has occurred. When connectivity is restored, the authorization will be automatically forwarded to the host for approval. This functionality is only applicable to Credit Card transactions, and only those that do not require Online PIN as the cardholder verification method.

Stand-In is primarily designed for short-term outages where real-time approvals are necessary to the operation of a business. It is disabled by default due to the liability/risk of use.

A stand-in approval puts liability for the transaction on the merchant. A stand-in approval does not provide any guarantee the merchant will receive the funds. The issuer may decline for a

number of reasons such as, Insufficient Funds, or a closed card account being presented. Use stand-in processing at your own risk!

There are a number of configuration options that allow a merchant to specify criteria for which transaction can be stored. Options such as, number of transactions currently offline, transaction amount, aggregate approved amount. Please see the `uniterm.ini` Configuration File section for additional configuration parameters.

If stand-in approves a transaction when the host is offline, the returned TTID will be prefixed with a `'U'`. This indicates it is a UniTerm TTID and is associated with the UniTerm instance that returned the approval. This TTID cannot be used with any other system. The TTID can be used with UniTerm reports to determine the transaction response once forwarded. At which point the real TTID will be returned. A response key of `u_standin=yes` will also be returned if a transaction was a stand-in.

Stored transactions will automatically attempt to be sent online for authorization every 15 minutes. If successfully sent the response will be recorded and can be accessed later by referencing the UniTerm TTID. Responses will be stored for a configurable number of days before being purged.

> Note: Store and Forward, or Stand-in processing requires additional server-side licensing. There is a Store and Forward license available for Monetra which allows all registered UniTerm instances to perform stand-in operations. Otherwise, stand-in processing will consume CardShield device licenses per UniTerm instance.

> Note: Store and Forward, or Stand-in processing was introduced as of UniTerm v9 as an optional feature.

2.4.1 Stand-In eligibility rules

Not all transactions qualify for stand-in processing. Below you will find a quick reference to assist in determining why a transaction may not be eligible.

Applicable to all methods of entry:

- Only cards able to be processed using credit card networks are eligible by default. E.g. not private label gift, EBT, or Pin-Debit. However, as of UniTerm v9.1, it is possible to allow Pin-Debit transactions to be stored by setting the corresponding flag on the Payment Server configuration.
- Missing or malformed authorization data (does not pass local edit checks)

EMV-specific failure reasons:

- Online PIN was attempted (unless corresponding flag set in the Payment Server configuration to allow this)
- ODA (Offline Data Authentication) was not performed (ignored for US Debit)
- ODA (Offline Data Authentication) static or dynamic failed
- Card on terminal exception file
- CDA (Offline combined dynamic data authentication with application cryptogram) failed

---

- SDA (Static data authentication) was selected
- Card application expired
- Card application not yet effective
- Cardholder verification was not successful
- Unrecognized Cardholder Verification Method (ignored for US Debit)
- PIN failures such as: limit try exceeded, pin entry device not present or inoperable, pin not entered (ignored for US Debit)
- Consecutive offline limit exceeded

2.5 ChipTab® - EMV Bar Tab support

Tabs are used to capture card data for later processing. This is for a single transaction and is not equivalent to tokenizing a card.

Typical use is at a bar where the customer opens a tab and as they order drinks or food the amount they're spending is added to the tab. Once they're finished and ready to leave the tab is closed and their card is charged.

This is commonly used in situations where payment is not taken at the same time as the product is given to the customer. And when there is the potential for the customer to leave without paying. Unlike a restaurant a bar can have a patron become intoxicated and forget to pay. In this situation the store has already captured the customer's card data and can close the tab against the card.

Tabs are intended to be opened and closed on the same day. Usually within a few hours. They are not intended for taking payment and shipping goods at a later time. Open tabs are auto purged if older than 14 days.

When closed the transaction data is sent to the Payment Server for authorization. If there is a communication error between UniTerm and the Payment Server or between the Payment Server and the processor tabs may be eligible for stand-in authorization. Stand-in authorization must be enabled and the transaction must be eligible for offline storage to take place. Tabs that are converted to offline authorizations will skip stand-in amount and count limits in order to prevent losing tabs.

Allowed card types that can be used for tabs:

- Credit
- Debit
- EBT Cash Benefits

These card types are specifically not allowed for tabs:

- Gift
- EBT Food Stamps

Note: ChipTab requires additional server-side licensing. The same Store and Forward license as required for Stand-in processing is also required for ChipTab support. If already performing stand-in, then no additional licensing is necessary.

Note: ChipTab was introduced as of UniTerm v9.

2.5.1 ChipTab eligibility rules

Not all transactions qualify for ChipTab processing. Below you will find a quick reference to assist in determining why a transaction may not be eligible.

Applicable to all methods of entry:

• Private Label Gift cards are not allowed
• EBT Food Stamps are not allowed

EMV-specific failure reasons:

• ODA (Offline Data Authentication) static or dynamic failed
• Card on terminal exception file
• CDA (Offline combined dynamic data authentication with application cryptogram) failed
• SDA (Static data authentication) was selected
• Card application expired
• Card application not yet effective
• Cardholder verification was not successful
• Unrecognized Cardholder Verification Method
• PIN failures such as: limit try exceeded, pin entry device not present or inoperable, pin not entered
• Consecutive offline limit exceeded

3 UniTerm Integration and Deployment Overview

3.1 Deployment

UniTerm is provided to developers during the integration process as an OS-specific standalone installer available from https://download.monetra.com/uniterm/production.

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

It is also acceptable to redistribute the standalone installation package.

For Android deployments, the UniTerm app is available via the Google Play Store, and it may be installed from there without a distribution license for Android versions 6 and higher. Alternatively, by obtaining a distribution license, the UniTerm software can be bundled with the application software as a Service component supporting Android versions 4.4 and higher.

For Apple iOS deployments, there are two distribution methods. The first is a framework that can be embedded into an application. Note that for iOS apps using this framework to be accepted into the App Store, Apple as well as the device manufacturers must provide approval for each type of BlueTooth device that will be used with the application. The second distribution method involves installation of the UniTerm application from the iOS App Store and using URL schemes for Inter-Process communication. Those interested in the framework for iOS should contact their sales representative.

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

Note: On Operating Systems that support package and/or executable signing, such as on Windows, MacOS, iOS and Android, the available packages are signed with an appropriate trusted distribution key by Monetra Technologies, LLC. Do NOT continue any installation if the package fails the automatic signature verification check. All downloads should also always be obtained from our secure HTTPS server at https://download.monetra.com/uniterm/ production and should not be downloaded if the browser displays a warning that the site certificate cannot be verified.

Note: UniTerm, as of v9, requires connectivity to a Payment Server powered by Monetra v8.7 or higher, such as provided by a private installation or a hosted gateway such as TranSafe.

Note: Notifications for new versions including feature, bugfix, and security release can always be found on the official Monetra Technologies RSS feed available at https:// www.monetra.com/release-notes/rss. There is no built-in upgrade process, each new version is released as a full package that overwrites the existing installation.

3.2 Versioning

Note: UniTerm, as of v9, requires connectivity to a Payment Server powered by Monetra v8.7 or higher, such as provided by a private installation or a hosted gateway such as TranSafe.

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 `9.Y.Z`. The major (`X`) version number component is fixed at `9`, 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 Payment Server 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 the Payment Server 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 the Payment Server. 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, the Payment Server will register this unique device id if a UniTerm license slot is available, otherwise the Payment Server will reject the registration request and UniTerm will cancel the transaction.

3.3.2 Device Definition

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

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

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

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

3.3.3 Management

Since UniTerm licensing is managed at the Payment Server server, all license administration (view licenses, delete licenses, etc) can be performed using either the Monetra Administrator GUI or via the Payment Server 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 the Payment Server, then the license slot is not eligible to be re-used for 7 days. However, if the same [deleted] device is re-presented, it can immediately re-consume the license slot.

3.4 Starting UniTerm

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

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

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

3.4.1 Command Line Options

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

- `-c` - Full path to the `ini` file to read. If not specified, it searches for the `uniterm.ini` in the paths documented in Section 4.1.
- `-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 Communication to UniTerm from Integration

The communication protocol for UniTerm is very similar to that of the Payment Server. 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 the Payment Server for processing.

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

Note: The communication channel with UniTerm can never transmit or receive sensitive card holder data, nor can it control or manage access to such data.

3.7.1 Network Communication

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

Normally, UniTerm listens on localhost on port 8123, and as of UniTerm v8.2, listens on both IPv4 and IPv6 if available. It is possible to make UniTerm accept connections from remote machines by configuring 'localonly=no' in the uniterm.ini. In order to use requests that do not normally require authentication, you must also configure 'sharedsecret=' in the uniterm.ini and send u_req_hmacsha256 in the request. Please see Section 5.2.5 for more information on shared secret usage.

3.7.2 Android Service Communication

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

3.7.3 Apple iOS

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

3.7.3.1 Framework

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

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

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

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

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

The "Actual used protocol(s)" listed above will be part of the `u_device` in the key/value pairs sent to UniTerm (prefixed with `MFi:` for `Made For iOS`).

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

3.7.3.1.1 APIs

Function	Description
`BOOL uniterm_initialize( void );`	Initialize UniTerm. This must be called at app startup and before any other UniTerm functions are called. It is recommended to put it in the `AppDelegate`'s `init` function.
`BOOL uniterm_modify_config( NSDictionary *req );`	Modify Configuration. The `uniterm.ini` file should not be edited directly by the app after initialization. Instead the

---

	uniterm_modify_config should be used. This ensures that any UniTerm cached network connections are properly updated. This function must be used because uniterm_run_trans is explicitly blocked from modifying the configuration. Direct modification of the uniterm.ini is possible before uniterm_initialize is called. Both uniterm.ini and uniterm-orig.ini need to be changed. Specifically uniterm-orig.ini should be a copy of the uniterm.ini.
NSDictionary *uniterm_run_trans(NSDictionary *req);	Main interface for running transactions with UniTerm. It accepts UniTerm key value pairs and runs the request. It returns a dictionary of the key value pair responses. This is a blocking function and should be used with dispatch_async. When the call completes it should notify the app that it is finished so the app can take appropriate action with the response.

3.7.3.2 URL Schemes

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

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

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

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

3.7.3.2.1 URL Scheme messaging format

The JSON format is an object with strings for the keys and values. If multiple requests are to be sent at once, each request object may be encapsulated in an array, with each member

---

in the array being a single request. When using request stacking, it is required that a unique `u_id` be passed with each request, which will be returned in the response and must be used for matching, never rely on the response array being in the same order as the request. If not using request stacking, `u_id` is not a required parameter.

Example JSON data:

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

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

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

Example Request:

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

Example Critical Error:

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

Example Successful Response:

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

3.8 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.

3.9 User Setup Permissions and Requirements

All authentication is managed by the Payment Server.

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

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

UniTerm also requires these permissions to operate:

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

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

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

3.10 Linux OS device access permissions

3.10.1 HID devices

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

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

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

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

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

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

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

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

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

3.10.2 Serial devices

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

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

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

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

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

3.11 Android - Embedding

3.11.1 HID Support

Android does not have blanket USB permissions. Access needs to be granted by the user on a per device basis. Permission granting is not supported by the UniTerm AAR and must be handled by the containing application itself. Once permissions are granted UniTerm can access the device.

The manifest must include the uses-feature for USB host.

```
<uses-feature android:name="android.hardware.usb.host" />
```

There are two methods for obtaining permissions. The Android USB Host documentation provides a detailed overview of the permission process: https://developer.android.com/guide/topics/connectivity/usb/host

UsbManager.hasPermission() should be used to determine if the app can access the device or if it needs permission from the user to do so. If the manifest method is used the request method may still be necessary to implement. However, the manifest method allows the user to associate the device with the application so permission only needs to be granted once.

3.11.1.1 Permission via Manifest

The device vendor and product ids can be registered by the Application though the manifest file. When the device is connected the user will be prompted if they want to open the device with the application. There is an option to always open with the given application the user can select. If they do not select a default application they will be prompted every time the device is connected. Once allowed the application can use the device.

The manifest will specify an intent filter for a given activity for USB device attached. A meta-data specifying supported devices is associated with the intent which Android uses to determine if the application supports the given device.

```
<activity ...>
  <intent-filter>
    <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
  </intent-filter>
  <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
    android:resource="@xml/device_filter" />
</activity>
```

The device_filter xml file specifying one or more devices using vendor an product ids. The ids must be a decimal number and cannot be hex.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <usb-device vendor-id="1234" product-id="5678" />
</resources>
```

The disadvantage of this method is it work off of the device being attached. If the application is running and the device is already connected the user will not be prompted.

3.11.1.2 Permission via Request Dialog

This method uses the `UsbManager.requestPermission()` function to display a permission request to the user to allow USB access for an already-connected device. The application will use an intent to make the request. A broadcast receiver will need to be registered with the intent in order for the application to receive the users response to the query. If approved by the user the application can use the device.

This does not grant access to USB in general. The device in question is part of the permission request. The user is only given permission for that specific device.

For this method the application should use the UniTerm request of `u_action=hidlist` to enumerate the currently connected devices and get the `path` column. Then use the path to get the relevant device handle out of `UsbManager.getDeviceList()`. This device can then be used for the permission request as per: https://developer.android.com/guide/topics/connectivity/usb/host#permission-d

3.12 Deploying UniTerm in a public-facing environment

In some environments it may be beneficial to deploy UniTerm such that it is "public-facing", meaning that untrusted (e.g. internet) users can access UniTerm. This is mostly useful for web-based POS systems which may need to talk to UniTerm via the back-end host rather than the front-end machine (while utilizing IP-enabled terminals). In this case, UniTerm may be running anywhere in the world, rather than on the POS system as is typically done.

There are a few requirements that must be followed to ensure UniTerm an sensitive data is protected and complies with the PCI-DSS requirements:

- Deploy UniTerm in a DMZ, behind a firewall with only specific ingress and egress ports allowed.

- Configure the use of external database, not the built-in default SQLite database.

- The database must reside within a different, private, system and security zone, and not be deployed in the DMZ with UniTerm.

- Ensure any terminals connecting via IP to UniTerm are using TLS v1.2 or higher, or are using P2PE encryption.

- The only egress ports that should be open for UniTerm are to the Payment Server server (by default port 8665) restricted to that host, and a similar rule allowing access to the external database.

- The only ingress ports should be the API port (by default 8123), and if using IP-enabled terminals, the port configured in the `[device_server]` section of `uniterm.ini`

4 Configuration

4.1 Configuration Files

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

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

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

- Windows:
  - `%APPDATA%/UniTerm/uniterm.ini`
  - same path as the `uniterm.exe` executable
- Mac OS X:
  - `~/Library/Application Support/UniTerm/uniterm.ini`
- Linux/Unix:
  - `~/.uniterm/uniterm.ini`
  - same path as the `uniterm` executable
- iOS/Android: N/A - embedded into the application bundle, meant to be modified via GUI or
  `u_action=modifyconfig`

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

4.2 Configuration Parameters

The parameters in this section are in standard `ini` format grouped by sections. Sections are
in the format of "`[section]`". The settings for each section are in key/value pair format of

---

"key=value". Each setting and section are delimited from each other using a new line, either Unix `LF` or Windows-style `CRLF` are acceptable.

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

4.2.1 Section: `[payment_server]`

- `host`: Required. Hostname/address where the Payment Server resides

- `port`: Required. Port to connect to the Payment Server on

- `ssl_cert_validate`: Optional. Controls validation of the SSL server certificate of the Payment Server. Possible Values are:
  - `full` - validate server certificate signature and require the full domain matches the certificate
  - `fuzzy` - validate server certificate signature and only the base domain matches the certificate
  - `validate` - validate only the server certificate signature, do not validate the domain name in the certificate
  - `none` - perform no server certificate validation
  If no value is specified, defaults to `full`. For self-hosted payment servers, a self-signed SSL certificate may be used. You must either explicitly deploy a signed certificate (signed by a trusted CA) with the Payment Server to be able to validate its certificate, or add the Payment Server's self-signed SSL certificate to the trust list (see `ssl_cadir`). It is required to use `full` if connecting to the Payment Server across the public internet.

- `ssl_cadir`: Optional. Path to a directory containing a list of the `PEM`-encoded trusted SSL Certificate Authority roots or individual server certificates to be added to the trust list. If available, the OS-provided trust list will be loaded first and any certificates in the provided directory will be appended. Only used when `ssl_cert_validate` is set to a value other than `none`.

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

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

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

4.2.2 Section: `[uniterm]`

4.2.2.1 Operating Parameters

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

- `sharedsecret`: Optional. The value specified is the shared secret to use for the communication protocol as part of an HMAC-SHA256 authentication token sent with the request for otherwise unauthenticated actions. It is required to use a shared secret for remote connections (`localonly=no`) if unauthenticated requests need to be used. Authenticated requests such as `u_action=txnrequest` do not use the shared secret, even if originating outside the local machine. Please see the Shared Secret section for more information.

  Authenticated requests are those requests that do not take a username or password which is authenticated by the Payment Server server.

  Note: `u_action=modifyconfig` requests use the `sharedsecret_mc` secret instead.

- `sharedsecret_mc`: Optional. Same as `sharedsecret` but used specifically for `u_action=modifyconfig` requests.

- `localonly`: Optional. If not specified, defaults to yes. If set to `no`, a `sharedsecret` must be set to allow access to otherwise unauthenticated requests.

- `ssl_cert`: Optional. SSL/TLS certificate. If not specified attempts to locate `ssl.crt` in the same path as `uniterm.ini`. If using the server needs to support multiple hostnames (as is supported via `SNI`), separate each certificate path with a semi-colon and it will load them in order. The order must match the order in `ssl_key`.

- `ssl_key`: Optional. SSL/TLS certificate private key. If not specified attempts to locate `ssl.key` in the same path as `uniterm.ini`. If using the server needs to support multiple hostnames (as is supported via `SNI`), separate each private key path with a semi-colon and it will load them in order. The order must match the order in `ssl_cert`.

- `ssl_cadir`: Optional. Enables verification of certificate presented by a remote client connection. Specifies a directory containing PEM files used to validate all clients connecting to UniTerm.

  Note: UniTerm is designed to keep the POS away from the cardholder data, most customers will not have a need for client certificate verification.

- `ssl_protocols`: Optional. Set the SSL/TLS protocols allowed for inbound connections to UniTerm (out of scope for PCI as no Card Holder Data is ever transmitted or managed by this connection).

  UniTerm defaults to secure settings, and this should not be changed unless necessary as it could have security and PCI implications.

  The value for this field is a space separated list of protocols. Valid protocols allowed to be configured are: `tlsv1.0, tlsv1.1, tlsv1.2` If the protocol is appended with a plus

---

(+) sign, then it means that protocol version or higher, for instance, `"tlsv1.1+"` implies `"tlsv1.1 tlsv1.2"`.

The default value is `tlsv1.0+`, however if `tlsv1.2` or higher is supported by integrators, it is recommended to change this to `tlsv1.2`

- `ssl_ciphers`: Optional. Set the SSL/TLS ciphers allowed for inbound connections to UniTerm.

  UniTerm defaults to a known secure cipher list, and this default list may change from release to release as the security environment changes. It is not recommended to set this parameter as it could have security and PCI implications.

  The value is a standard OpenSSL cipher string.

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

- `idle_conn_timeout`: Optional. Once a POS connection to UniTerm has been idle for the specified amount of time in seconds, it will be closed. Default value is 30, use 0 to disable.

- `req_receive_timeout`: Optional. The maximum amount of time in seconds between UniTerm receiving data and fully parsing the request. As soon as data is received the timer will start. Once a full request has been received the timer will stop. If the time is exceeded the connection is dropped. Default value is 4, use 0 to disable.

- `req_max_size`: Optional. Maximum number of bytes allowed for a single request. This is _not_ bytes allowed per connection. A single connection can send multiple requests but a single request cannot exceed this limit. Default is 20971520 (20MB).

Note: While the vast majority of requests will be very small, this needs be large enough to handle device firmware upgrades.

- `max_conns`: Optional. Maximum number of open connections. If the limit is exceeded new connects will not be accepted until the total number of open connections drop below the limit. High volume systems should increase this limit appropriately for the system's resources. Default value is 1000.

- `password_iterations`: Optional. Number of PBKDF2 iterations performed when hashing passwords. The larger this number the longer password verification will take. This can cause system slowdown when used with a default (SQLite) database. It is not recommended to exceed a count of 1,000,000 when using SQLite. Default value is 10000.

4.2.2.2 Feature Parameters

- `idle_message`: Optional. Set the default idle message displayed on any device when not processing a transaction. This can be overwritten on a per-device level using the `u_deviceidlemessage` parameter in the protocol. This is not supported on all devices.

- `unsupportedcard`: Optional. If not specified, defaults to not supported. Allowed values are `txnreq`, `cardreq`. The values may be specified in a pipe-delimited format to include more than one. For legacy configurations, a value of `yes` is an alias for `txnreq|cardreq`, and a value of `no` is the equivalent of blank or not set. `txnreq` allows the `u_action=txnrequest` to support non-financial cards, and `cardreq` allows the use of the `cardrequest` functionality.

  This allows trackdata to be returned to the caller for `txnrequest` and `cardrequest` only when the card type is confirmed to be non-financial. This is to allow in-store private-label gift (on `txnrequest`) as well as manager cards. The card must be returned unencrypted from the reader to be supported.

- `nocvmfloor`: Optional. If not specified, defaults to disabled, should be specified as a dollar amount. This configuration value will disable cardholder verification (e.g. PIN or Signature) when the transaction amount is less than this limit. For instance if the value is set to 50.00, and a 40.00 authorization is attempted as a swipe transaction, they will NOT be prompted to sign, however a 60.00 authorization would be prompted to sign. This feature works with EMV contact as well, however EMV Contactless has its own set of limits advertised by the Payment Server that UniTerm will honor. Applies only to Credit Card purchases.

- `guimode`: Valid options are `normal` and `touchscreen`. A modifier value of `fullscreen` can also be added to force the GUI to be rendered full screen rather than in a window, the modifier will be separated from the mode by a pipe (|), for example `touchscreen|fullscreen`. If no `guimode` is specified, it defaults to `normal`. TouchScreen mode enlarges all text in dialogs and provides an on-screen keypad to be used for manual card entry. Only numeric input is allowed in `touchscreen` mode, if an alpha-numeric Postal code needs to be entered (such as for Canada), a keyboard must be used. Normal mode is designed for use at a workstation with a keyboard and mouse.

- `returnbin`: Optional. Boolean, defaults to no. If set to yes, this allows UniTerm to return the first 6 digits of the card number as part of the response for `txnstart`. This data can then be used for `BIN` checking to determine additional card classification such as if it is a health benefits card or qualified Visa card for Debt Repayment.

Note: Only the last 4 digits of the PAN are allowed to be printed on receipts. It is the integrator's responsibility if using this feature to ensure the first 6 digits are masked on receipts.

- `cardwait`: Optional. `Txnstart` card presentation timeout in seconds. `-1` uses internal timeout as determined by the device (default). `0` waits indefinitely. Any other value is the timeout specified in seconds.

  For devices that support specifying a card presentation timeout, this value will control how long until a timeout occurs. This is mostly useful in unattended/kiosk environments where you may want to prompt indefinitely waiting on an order. This is only applicable to the `txnstart` method when the order amount is not yet known. Not all devices support overriding their internal timeout, and it is up to the POS to handle this situation when an unexpected timeout occurs.

- `signaturetimeout`: Optional. Action to take when signature entry times out.

  If a customer does not sign, the transaction is in an undefined state. The customer could have refused to sign because they no longer want to continue with the transaction. Or they could have forgotten to sign.

  In the former it's ideal to treat this condition as a cancellation and automatically reverse the transaction. In the latter it is ideal to honor the auth and request that the customer sign the receipt.

  Choices:
  - reverse - Default. Will reverse the transaction.
  - ignore - Will ignore the timeout. `u_need_signature=yes` and `u_signature_timeout=yes` will be returned to the POS.

4.2.3 Section: `[device_server]`

UniTerm can function as a server with remote devices initiating inbound connections, some call this "client mode". Not all devices support such a feature. Connected devices are addressed via serial number rather than ip address.

- `ipserver=u_devicetype:port`: Start an unencrypted IP listening port for the specified device type and port. If multiple device types or multiple ports are desired, can be specified using a semi-colon separated list. Not all device types are supported.

  Example: `ipserver=ingenico_rba:6000;ingenico_rba:9000`

- `ipserver_standalone=u_devicetype:port`: Start an unencrypted IP listening port for the specified device type and port, which will start a flow for a standalone terminal (operation via device menus only, no API operation). If multiple device types or multiple ports are desired, can be specified using a semi-colon separated list. Not all device types are supported.

  Example: `ipserver_standalone=ingenico_rba:6002;ingenico_rba:9002`

- `sslserver=u_devicetype:port`: Start a SSL/TLS listening port for the specified device type and port. If multiple device types or multiple ports are desired, can be specified using a semi-colon separated list. Not all device types are supported.

  Example: `sslserver=ingenico_rba:6000;ingenico_rba:9000`

- `sslserver_standalone=u_devicetype:port`: Start a SSL/TLS listening port for the specified device type and port, which will start a flow for a standalone terminal (operation via device menus only, no API operation). If multiple device types or multiple ports are desired, can be specified using a semi-colon separated list. Not all device types are supported.

  Example: `sslserver_standalone=ingenico_rba:6003;ingenico_rba:9003`

- `ssl_cert=[filepath]`: Location of a SSL/TLS server certificate to present to client upon connection. Required when using `sslserver`

- `ssl_key=[filepath]`: Location of a SSL/TLS server private key associated with the `ssl_cert`. Required when using `sslserver`

- `ssl_cadir=[path]`: Optional. Client certification verification. Only clients connecting with a certificate signed by these Certificate Authorities will be allowed to connect. If a directory is not provided, will not attempt client certificate verification.

- `ssl_protocols`: Optional. Set the SSL/TLS protocols allowed for inbound connections to UniTerm from a device.

  UniTerm defaults to secure settings, and this should not be changed unless necessary as it could have security and PCI implications. The value of this should only be lowered if the device communication does not traverse the public internet or `P2PE` is being utilized.

  The value for this field is a space separated list of protocols. Valid protocols allowed to be configured are: `tlsv1.0, tlsv1.1, tlsv1.2` If the protocol is appended with a plus (+) sign, then it means that protocol version or higher, for instance, `"tlsv1.1+"` implies `"tlsv1.1 tlsv1.2"`.

  The default value is `tlsv1.2`

- `ssl_ciphers`: Optional. Set the SSL/TLS ciphers allowed for inbound connections to UniTerm from a device.

  UniTerm defaults to a known secure cipher list, and this default list may change from release to release as the security environment changes. It is not recommended to set this parameter as it could have security and PCI implications.

  The value is a standard OpenSSL cipher string.

- `max_conns`: Optional. Maximum number of open connections for each server. If the limit is exceeded new connects will not be accepted until the total number of open connections drop below the limit.

  Each server is treated independently for the total count. For example, `ingenico_rba:6001;ingenico_rba:9001` port 6001 will allow up to `max_conns` and port 9001 will allow up to `max_conns`. Giving a total of `max_conns` * 2 allowed connections.

  High volume systems should increase this limit appropriately for the system's resources.

If not set will default to UniTerm's `max_conns` config value. If neither are set default is 1000. This acts as an override.

4.2.4 Section: `[db]`

UniTerm uses a database to store state data. If no database is configured, it will automatically default to an `sqlite` database in a subdirectory named `data` of the directory containing the `uniterm.ini`.

- `type`: Database type to use:
  - `sqlite` - Default, server-less.
  - `mysql`
  - `postgresql` - Linux/Unix only, use `ODBC` for Windows
  - `oracle` - Linux/Unix only, use `ODBC` for Windows
  - `odbc` - for Microsoft Windows, `iODBC`, or `UnixODBC`
  - `db2` - Linux/Unix only, for direct `DB2` connectivity

- `connection`: A DB-specific connection string or `DSN`. This string often configures the host/port, and available options for the driver in use. The connection strings are a set of key/value pairs, with keys separated from the values with an equal sign (=), and values separated by a semi-colon (;). If quoting is in use, a single-quote (') is recognized, and an escape character of a backslash (\\) can be used. E.g.: `host=10.130.40.5:3306;ssl=yes`

  `sqlite` options:
  - `path`: Required. File system path to SQLite database.
  - `journal_mode`: Optional. Defaults to `WAL` if not specified, other options include `DELETE`
  - `analyze`: Optional. Defaults to "TRUE" if not specified. On first connect, automatically runs an analyze to update index statistics if set to "TRUE".
  - `integrity_check`: Optional. Defaults to "FALSE" if not specified. On first connect, automatically runs an integrity check to verify the database integrity if set to "TRUE".
  - `shared_cache`: Optional. Defaults to "FALSE" if not specified. Enables shared cache mode for multiple connections to the same database.
  - `autocreate`: Optional. Defaults to "TRUE" if not specified. The default is to auto-create the database if not found, set this to "FALSE" to error if the database does not exist.

  `mysql` options:
  - `db`: Required. Database Name.
  - `socketpath`: Conditional. If using Unix Domain Sockets to connect to MySQL, this is the path to the Unix Domain Socket. Use the keyword of 'search' to search for the socket based on standard known paths. Cannot be used with host.
  - `host`: Conditional. If using IP or SSL/TLS to connect to MySQL, this is the hostname or IP address of the server. If not using the default port of 3306, may append a ":port#" to the end of the host. For specifying multiple hosts in a pool, hosts should be comma delimited. Cannot be used with `socketpath`. E.g: `host=10.40.30.2,10.50.30.2:13306`
  - `engine`: Optional. Used during table creation, defaults to `INNODB`. The default data storage engine to use with mysql. Typically it is recommended to leave this at the default.
  - `charset`: Optional. Used during table creation, defaults to `UTF8`.

- `max_isolation`: Optional. Sets the maximum isolation level used for transactions. This is used to overwrite requests for `SERIALIZABLE` isolation levels, useful with Galera-based clusters that do not truly support `Serializable` isolation. Should use "SELECT ... FOR UPDATE" type syntax for row locking. Available settings: `"REPEATABLE READ"`, `READ COMMITTED"`

`postgresql` options:

- `db`: Required. Database Name.
- `host`: Required. This is the hostname or IP address of the server. If not using the default port of 5432, may append a ":port#" to the end of the host. For specifying multiple hosts in a pool, hosts should be comma delimited. E.g: `host=10.40.30.2,10.50.30.2:15432`
- `application_name`: Optional. Application name to register with the server for debugging purposes.

`oracle` options:

- `dsn`: Conditional. Data Source Name as specified in `tnsnames.ora`, or a fully qualified connection string. If not specified, both host and `service_name` must both be specified and a connection string will be dynamically generated. Use of this parameter negates the ability to use load balancing and failover logic, but facilitates the use of Oracle's equivalent functionality. An example of a fully qualified connection string would be:

```
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(Host = 10.100.10.168)(Port = 1521))
  (CONNECT_DATA = (SERVICE_NAME = orcl))
)
```

- `host`: Conditional. If `dsn` is not specified, this parameter must be specified along with the `service_name` parameter. This is the hostname or IP address of the server. If not using the default port of 1521, may append a ":port#" to the end of the host. For specifying multiple hosts in a pool, hosts should be comma delimited. Cannot be used with `dsn`. E.g: `host=10.40.30.2,10.50.30.2:11521`
- `service_name`: Conditional. If `dsn` is not specified, this parameter must be specified along with the host parameter. Cannot be used with `dsn`. E.g: `service_name=orcl`

`odbc/db2` options:

- `dsn`: Required. Database Source Name.
- `mysql_engine`: Optional. Used during table creation when the underlying database is MySQL, defaults to `INNODB`. The default data storage engine to use with mysql. Typically it is recommended to leave this at the default.
- `mysql_charset`: Optional. Used during table creation when the underlying database is MySQL, defaults to `UTF8`.

- `ro_connection`: Optional. Route read-only requests to a different database pool of nodes. Same configuration syntax as `connection`.

- `conn_flags`: Optional. These control how the connection is handled. Settings are a set of flags, delimited with pipes (|).
  - `prespawn`: Pre-spawn all connections, not just the first. Without this, the remaining connections are on-demand.

- `no_autoretry_query`: If a non-transactional query is rolled back due to a deadlock or connectivity failure, the default behavior is to automatically retry the query, indefinitely. For queries executed as part of may be dependent on prior queries in the transaction. This flag will turn off the auto-retry logic. NOT RECOMMENDED.
  - `load_balance`: If there are multiple servers specified for the connection string, this will load balance requests across the servers instead of using them for failover.
  E.g.: `conn_flags=prespawn|load_balance`

- `num_conns`: Optional. Maximum number of SQL connections to attempt to create. Valid range 1-1000. Default 2.

- `num_ro_conns`: Optional. Maximum number of SQL connections to attempt to create for the read only pool. Valid range 1-1000. Default 2.

- `username`: Conditional. For databases which require authentication, the username.

- `password`: Conditional. For databases which require authentication, the password.

- `to_reconnect`: Optional. How many seconds to allow a connection to be used before a disconnection is forced. The connection will be terminated even if not idle, termination will occur when a connection is returned to the pool instead of prior to use to prevent unexpected delays. This can be used to either redistribute load after a node failure when load balancing, or to fall back to a prior host. Set to 0 for infinite. Default is 0.

- `to_idle`: Optional. Maximum amount of time a connection can have been idle to be used. Some firewalls may lose connection state after a given duration, so it may be advisable to set this to below that threshold so the connection will be forcibly terminated rather than use. The connection will be terminated before use and the consumer will attempt to grab a different connection from the pool, or start a new one if none are available. Set to 0 for infinite. Default is 0.

- `to_fallback`: Optional. Number of seconds when a connection error occurs to a host before it is eligible for "fallback". If this isn't set, the only time the first host will be re-used is if the secondary host(s) also fail. This should be used in conjunction with reconnect_time_s. Set to 0 to never fallback. Not relevant for load balancing, the host will always be in the attempt pool. Default is 0.

4.2.5 Section: `[logging]`

UniTerm v9 introduces logging support with user-definable levels. No levels on production builds can output any sensitive data. Available configuration parameters:

- `level`: Pipe delimited list of log levels that are used to determine the data written to the log file. Available levels are:
  - `error`: Errors
  - `warn`: Warnings
  - `info`: Informational Messages
  - `conn`: Inbound connection status
  - `tran_detail`: Request/Response parameters for inbound connections
  - `mon_detail`: Request/Response parameters outbound to the Payment Server

---

The default value is: `info|error|conn|warn|tran_detail|mon_detail`

- `system`: Which logging infrastructure to use. Multiple infrastructures can be used simultaneously if specified as pipe delimited.
  - `syslog`: Local syslog, only available on Unix systems
  - `tcpsyslog`: TCP syslog, available on all systems to send to remote syslog server via TCP.
  - `file`: Log to local file.

  The default value is `file`.

- `time_prefix`: Timestamp prefix to add to each line of the log specified as a format string. Options:
  - `%t` - Unix Timestamp
  - `%M` - 2-digit Month
  - `%a` - abbreviated month (e.g. Jan)
  - `%D` - 2-digit day of month
  - `%d` - abbreviated day of week (e.g. Sun)
  - `%Y` - 4-digit year
  - `%y` - 2-digit year
  - `%H` - 2-digit hour
  - `%m` - 2-digit minute
  - `%s` - 2-digit second
  - `%u` - microseconds
  - `%z` - time zone offset

  The default is `%a %D %H:%m:%s.%u %z` which may generate a line such as:

```
Jan 11 09:19:11.426235 -0500 [INFO]:
```

- `syslog_facility`: The syslog facility to log to, used for both TCP syslog and system syslog. Defaults to `user` if not set. May also use `daemon` or `local0-7`

- `tcpsyslog_host`: Host or address of the TCP syslog server to log to.

- `tcpsyslog_port`: TCP port the TCP syslog server is listening on.

- `tcpsyslog_bufsize`: Maximum KB that the message queue will use when the remote server is offline. Use 0 for unlimited, use with caution. Default is 512.

- `file_directory`: Directory where log files should be kept when using file logging. Files will be called `uniterm.log[.#]`. The default is in a subdirectory named `log` in the same directory as the `uniterm.ini`.

- `file_keep`: Number of rotated log files to keep. Defaults to 10.

- `file_archive`: Command to be run against file on rotation. Default is nothing.

- `file_archive_ext`: If `file_archive` changes the file name, such as after compressing, adding a `.bz2` extension, it must be configured here so rotated files can be found.

- `file_rotate_autostart`: Boolean field for whether or not to automatically initiate a rotate on startup. Default is yes.

- `file_rotate_days`: Number of days after log creation that the log file should be rotated. Use 0 to disable. Defaults to 7.

- `file_rotate_size`: Size in KB to automatically rotate log file. Defaults is 10240 (10MB). Use 0 to disable.

4.2.6 Section: `[tab]`

UniTerm v9 introduces tab support for capturing a customers card before charging it such as is used by bars. Available configuration options:

- `only_standin`: Only allow the same cards that are allowed for stand-in/store-and-forward processing to be used for tabs. This should be used if also using stand-in processing to ensure only transactions that can be approved as stand-in will be used for tabs. Otherwise cards that cannot be used for stand-in processing will decline if there is a connectivity failure. Default value is no.

4.2.7 Section: `[standin]`

UniTerm v9 introduces stand-in/store-and-forward support for locally approving transactions when connectivity is unavailable. Available configuration options:

- `enabled`: Whether or not stand-in processing should be enabled. Default value is no.

- `auth_code`: Authorization code to send back for a stand-in approval, dynamically generated based on provided pattern. Default value is `SA!!!!`.

  Special symbols can be used to generate random authorization codes.
  - `!` = Number 0-9
  - `$` = Alpha uppercase
  - `&` = Number 0-9 or alpha uppercase
  All other characters are used as as literals.

  If using a generated sequence it's recommended to use a prefix to denote a stand-in authorization that includes a non-numeric non-alpha character to differentiate from 6 digit alpha numeric online authorization codes.

  A static code can also be used if dynamic codes are unwanted.

  Field has a maximum of 6 characters. If longer will truncate on the left.

  Examples:
  - `SA!!!!` -> `SA8526`
  - `SA$$$$` -> `SAZHFW`
  - `SA&&&&` -> `SA9GS7`
  - `SNF999` -> `SNF999`

- `verbiage_approve`: Verbiage returned for stand-in approval. Default value is `STANDIN AUTH`.

- `verbiage_decline`: Verbiage returned for stand-in decline. Default value is `STANDIN DENY`.

- `max_trans`: The maximum number of transactions that can be stored at any given time. Applies to each individual user. Is NOT allowed to be 0, if set to 0 (or not set) stand-in will be disabled.

- `max_tran_amount`: Maximum single transaction amount eligible for offline approval. Is NOT allowed to be 0.00, if set to 0.00 (or not set) stand-in will be disabled.

- `max_aggregate_amount`: Maximum aggregate transaction amount eligible for storage. Applies to individual users. Is NOT allowed to be 0.00, if set to 0.00 (or not set) stand-in will be disabled.

- `max_offline_time`: Maximum number of days UniTerm can be offline before it will reject offline approvals. There is a hard maximum of 8 days allowed. Is NOT allowed to be 0 it must be configured appropriately, if set to 0 (or not set) stand-in will be disabled.

- `upload_interval`: How often to attempt to upload stored transactions in minutes. Transactions are sent one at a time across all users. Transactions are only sent if the user is not currently processing any transactions. If the given user is processing transactions that user's stored transactions will be skipped and will attempt to be sent during the next upload interval. Default is 15.

- `keep_response_days`: Number of days to keep responses to stored transactions. Stored transaction responses older than the configured number of days will be deleted. At which point there is no way to associate the UniTerm TTID returned with a stand-in approval to the actual TTID generated when the transaction is sent. Default is 30.

4.2.8 Section: `[blacklist]`

UniTerm v9 introduces automatic intrusion detection and prevention that is configurable to block potentially malicious requests. All events are tracked per originating IP address. Available configuration options:

- `time_span`: Time in minutes of the sliding window used to determine if successful events should cause a ban. Use 0 to disable. Default value is 2.

  `time_span_fail`: Time in minutes of the sliding window used to determine if failure events should cause a ban. Use 0 to disable. Default is 5.

  `ban_time`: Time in minutes the ban should remain in effect. Use 0 to disable. Default is 15.

  `ban_extend`: When a client connects after it's been banned the ban expiration time will be extended by this number of minutes. Use 0 to disable. Default is 10.

  `whitelist`: Comma delimited list of subnets that will never be banned. Default is all local subnets: `127.0.0.0/8,192.168.0.0/16,10.0.0.0/8,172.16.0.0/12,::1/128,fc00::/7`

  `max_connect`: Maximum number of repeatedly open successful connections in `time_span` window. Use 0 to disable. Default is 40.

`max_requests`: Maximum number of successful requests, excluding `ping` and `status` in `time_span` window. Use 0 to disable. Default is 40.

`max_failures`: Maximum number of failure events (e.g. Invalid `u_action`, message parse failure, authentication failure) within the `time_span_fail` window. Use 0 to disable. Default is 5.

`max_ping`: Maximum number of successful protocol-level ping requests allowed within `time_span` window. Use 0 to disable. Default is 1500.

`max_status`: Maximum number of successful status requests allowed within `time_span` window. Use 0 to disable. Default is 1000.

`max_action`: Maximum number of a given action. This differs from `max_requests` in the fact that this is tracked per-action, rather than grouping all actions together. Use 0 to disable. Default is 20.

5 UniTerm Protocol

5.1 Overview

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

UniTerm is self-contained and does not require any external operating system dependencies or third party dependencies, it is only reliant on the Operating System and supplied network infrastructure (it does not depend on any OS-provided cryptographic subsystems).

By default UniTerm listens on port 8123 and connects to the configured remote instance of the Payment Server on port 8665. These ingress and egress ports and hosts are configurable as per Section 4.2.2.1 and Section 4.2.1.

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

---

Monetra Client Interface Protocol Specification to come up with a complete list of parameters necessary to complete transactions.

Note: Referenced documentation available at https://www.monetra.com/developers:

- Monetra Client Interface Protocol Specification
- Monetra IP, SSL, and Drop File Specification (only TLS/SSL supported by UniTerm)
- Monetra XML Specification

5.2 UniTerm Request Parameters

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

PARAMETER	VALUE
username	The Payment Server username to authenticate as. For security reasons this should be a restricted subuser account.
password	The Payment Server password associated with username.
u_action	See UniTerm Actions (`u_action`) for a complete description of each action. • deviceload • txnrequest • txnstart • txnfinish • txnquick • devicepolemsg • cardrequest • passthrough • cancel • deviceupload • deviceprint • devicereboot • reqsignature • reqconfirm • reqinput • devicetypes • status • seriallist • bluetoothlist • mfilist • blelist • hidlist • shutdown • modifyconfig • version • deviceinfo • standin

		• `tab` • `expirecached`

	`u_flags`	`Txnrequest`, `Txnquick`, `Txnstart`, `Txnfinish` and `Cardrequest` only. Multiple flags may be sent per data ticket request. All flags are separated by a pipe (|) symbol.
		
		 • `DEVICEONLY` - Suppresses display of clerk facing prompting and feedback, also known as GUI mode. For instance, on a swipe request, no swipe dialog would be presented nor would the clerk be informed of the status for customer-facing device interaction.
		
		     Note: Setting this mode will prevent keyboard emulation readers from working, it will also prevent the ability for accepting clerk keyed input via a keyboard. On Android, iOS, and UniTerm launched in console mode, this flag is automatically implied as they do not support a GUI mode of operation.
		
		 • `GUIONLY` - Suppresses use of a referenced physical device. This flag can be used to utilize the physical device's UniTerm license for a clerk-facing GUI request, without using the device itself for card input. Without this flag, the GUI would register an additional UniTerm license based on the machine's unique id.
		 • `KEY` - Perform capture of manually keyed data. Not valid on `cardrequest`. If an `account` or `expdate` field is passed in on the request to UniTerm, the fields will be pre-populated in GUI mode.
		
		     Note: If a manually keyed EBT card is to be entered for a Food Stamp (SNAP) transaction, pass the qualified food amount via `u_foodamount` in order to go through the EBT flow and PIN prompting. Do NOT pass `u_foodamount` if you are NOT intending to run a Food Stamp transaction as keyed.
		
		 • `AVS` - Request AVS data. Only allowed on keyed transactions. If a `street` or `zip` field is passed in on the request to UniTerm, the fields will be pre-populated in GUI mode.
		 • `CVV` - Request for CVV data. Only allowed on keyed transactions.
		 • `GIFTPIN` - Request a PIN for gift cards, when a gift card is presented. Does nothing if a different card type is presented.
		 • `NOTIP` - Disable tip prompting if enabled via the Payment Server merchant account configuration of `merch_tippercent`.

	• `NOCASHBACK` - Disable cash back prompting if enabled via the Payment Server merchant account configuration of `merch_cashbackmax`. • `NOCONFIRM` - When performing a QuickChip finalization via `txnfinish` do NOT prompt the cardholder to confirm the amount prior to running the authorization. • `NOSIGNATURE` - If using a signature-capable device, but an integrator does not want to use the auto-signature-capture capabilities in the UniTerm flow, this flag will treat the device as if it is not capable of accepting a signature. The `u_need_signature` response flag will be set appropriately for the transaction requirements. • `DELAYRESPONSE` - This flag will cause UniTerm to send the transaction response back to the caller after the device has been closed. The device can be used immediately for a subsequent transaction with this flag, otherwise device closing will happen in the background which will hold a lock on the device for at least 100ms, or possibly longer. • `CARDMETA` - This flag will cause UniTerm to request the card metadata directly from the Payment Server (v8.9.0+) rather than relying on less detailed internal information. The relevant metadata would be from a Global BIN table, which contains information such as if a card is Debit-capable, HSA/FSA capable, and so on. When used during a `txnstart` request, it allows an integrator to make decisions based on the card presented (such as for support of Debt Repayment or Healthcare transactions). When used during a `txnrequest, txnquick` request, it optimizes the flow on transactions such as not prompting for Debit on a swipe transaction if the card is not debit capable.
u_device	Requests: `Txnrequest, Txnquick, Txnstart, Cardrequest, DeviceInfo, DeviceLoad, ExpireCached, DeviceUpload, DevicePrint, DeviceReboot, ReqSignature, ReqConfirm, ReqInput` 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. • `HID[:serialnum]` - `HID` (such as `USB-HID`), takes an optional serial number • `SER:port` - Serial • `MFI:protocol,[serialnum]` - Made for iOS (MFi), as retrieved from `u_action=mfilist` • `BT:mac,[uuid]` - Bluetooth • `BLE:identifier` - Bluetooth LE using device identifier as retrieved from `u_action=blelist, u_blelist=scan`

	`BLESRV:service_uuid` - Bluetooth LE using the device service id as retrieved from `u_action=blelist` Note: Using the service id will connect to the first device connected exposing the service use the identifier method to use a specific device. • `IP:ipaddr:port` - IP • `IPCLIENT:serialnum` - IP Client Mode. Currently only supported for RBA devices support RBA client mode for ethernet connectivity. UniTerm will act as a server and the devices will connect to UniTerm and be referenced by their serial number.
u_devicetype	Requests: `Txnrequest, Txnquick, Txnstart, Cardrequest, DeviceInfo, DeviceLoad, ExpireCached, DeviceUpload, DevicePrint, DeviceReboot, ReqSignature, ReqConfirm, ReqInput` The unique device type supported by UniTerm as found via a `devicelist` request. Required if `u_device` is provided.
u_language	`Txnrequest`, `Txnquick` and `Txnstart` only. Used to override terminal defaults for display of text prompts. Current choices are "en" or fr".
u_deviceidlemessage	`Txnrequest, Txnquick, Txnstart`, or `deviceload` only. Sets a message that the terminal should display when idle, this will be persistently cached by UniTerm and associated with the device serial number.
u_forceload	`deviceload` only. Values allowed are `yes`, `no`, and `full`. If not specified defaults to `no`. A value of `yes` will force a reload of all EMV parameters even if UniTerm believes the device already has the latest set of parameters. A value of `full` will additionally force reloading of all other objects UniTerm maintains, including but not limited to, configuration values, forms, and images.
u_id	`Txnrequest, Txnquick, Txnstart, TxnFinish, Cardrequest, Status`, and `Cancel` only. A unique id (generated by the calling application) that identifies the transaction. This is used for checking the status or canceling the transaction. Without this id the transaction state cannot be queried.
amount	`Txnrequest, Txnquick`, and `TxnFinish` only. Transaction amount. Required.
u_b64data	`Deviceupload` only. Base64-encoded file data to upload. Required.
u_filename	`Deviceupload` only. Filename of the file to write. Required.

u_text	`Deviceprint` only. ASCII pre-formatted text to print to receipt printer. Required.
u_message	`Reqconfirm`, `Devicepolemsg` only. Textual request message to prompt user for confirmation. Required.
u_inputtype	`Reqinput` only. Type of user input being requested. Required. Possible values: • `PHONENUM` - Request user to input a phone number. This may be useful for loyalty card prompting. • `INVOICENUM` - Request user to input an invoice/order/ticket number • `ZIP` - Request user to input a zip code • `TIP` - Request user to input a tip amount. Requires an `amount` key value pair to be passed to calculate tip percentages. Uses the tip percentages configured within the Payment Server merchant account configuration.
u_foodamount	`Txnrequest`, `Txnquick`, `Txnstart`, or `Txnfinish` only. Dollar amount indicating amount of qualified food purchases for EBT Food Stamps (SNAP). Or during an `Txnstart` the food amount may not yet be known, but to enable EBT prompting, a special value of `maybe` may be passed, followed by the the real amount in `Txnfinish`. If the amount passed in `Txnfinish` is zero, but EBT Food Stamps was selected by the card holder, the transaction will be aborted.
u_req_hmacsha256	Required on unauthenticated requests when the `sharedsecret` configuration parameter is set. Please see Section 5.2.5 for more information.
u_req_sequence	Used in conjunction with `u_req_hmacsha256`. Please see Section 5.2.5 for more information.
u_req_timestamp	Used in conjunction with `u_req_timestamp`. Please see Section 5.2.5 for more information.
u_blelist	`Blelist` only. The only valid value to be passed to this is `scan`, which will list all supported present devices instead of just known device services.
u_rcpt	`txnrequest`, `txnquick`, `txnfinish`, `tab`, `standin`, `passthrough`, `passthroughmac` only. Whether a pre-formatted receipt should be added, and optionally the format to output the receipt in. May specify a pipe-delimited list of receipt types to output multiple receipt formats. Please see Appendix C for more information.
payment_server.host	`Modifyconfig` only. Modify the `[payment_server]` host configuration value.

---

payment_server.port	`Modifyconfig` only. Modify the `[payment_server]` port configuration value.
uniterm.idle_message	`Modifyconfig` only. Modify the `[uniterm]` `idle_message` configuration value.

5.2.1 UniTerm Actions (`u_action`)

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

u_action	AUTH?	description
deviceload	Yes	Load a device with EMV and/or Interac parameters. This request will start a terminal download of EMV and/or Interac parameters to load into the device being used. Requires `username`, `password`, `u_device`, and `u_devicetype` parameters. 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	Yes	Full Transaction Request for card holder data. Request a transaction be performed, all normal Payment Server transaction parameters should be included in the request that may be required for processing or to comply with interchange requirements (e.g. `action`, `username`, `password`, `amount`, `nsf`, `ordernum`, etc). Sensitive data, explicitly, should not be sent (e.g. `trackdata`, `account`, `cvv2`, `pin`) as that data will be retrieved by UniTerm either via the GUI or via a card entry device and forwarded to the Payment Server server as part of the transaction request. UniTerm will fully control the transaction flow and user prompting for such a transaction. This request should only be used when card data entry is required as part of the transaction, for other operations `passthrough` may be a more appropriate action.

`txnstart`	Yes	Transaction Start Request for card holder data. Request to start a QuickChip transaction, similar in function to a `txnrequest` message, but does not accept a transaction amount. Needs to be followed up with a `txnfinish` or `cancel` request. Once a card is presented and the cardholder has completed any necessary verification and removed their card, a response will be returned with card information metadata such as the card type, some basic EMV card parameters, and the last 4 digits of the card number. Though the purpose of this request is to support QuickChip, it still supports non-EMV transactions. A `u_id` unique parameter is required to be sent with this request for completing the transaction. See QuickChip for more information.
`txnfinish`	Yes	Transaction Finish Request. Request to finish a QuickChip transaction. Requires the integrator send the same `u_id` as was sent with the initial `txnstart` request. The integrator can choose to send additional parameters to pass through to Payment Server with this request based on information returned from `txnstart` (e.g. card type). Will return the same response parameters as `txnrequest`. See QuickChip for more information.
`txnquick`	Yes	Full Transaction Request for card holder data. Request a transaction be performed, all normal Payment Server transaction parameters should be included in the request that may be required for processing or to comply with interchange requirements (e.g. `action`, `username`, `password`, `amount`, `nsf`, `ordernum`, etc). Sensitive data, explicitly, should not be sent (e.g. `trackdata`, `account`, `cvv2`, `pin`) as that data will be retrieved by UniTerm either via the GUI or via a card entry device and forwarded to the Payment Server server as part of the transaction request. This operation is functionally similar to `txnrequest` however internally it follows a Quick Chip flow (like `txnstart` + `txnfinish`) which allows the cardholder to remove their card earlier. This may make a transaction appear to be faster to an end user.
`devicepolemsg`	Yes	Pole Display Update Message. Only valid after a Quick Chip (`txnstart`) message, before `txnfinish` is called. Requires the integrator send the same `u_id` as was sent with the initial `txnstart` request. Devices that support a pole display can have it updated until the transaction is completed. The message (`u_message`) must be pre-formatted for the proper width of the display.

cardrequest	Yes	Non-Financial Card Entry Request. UniTerm will prompt for card entry, and if it is determined the card is non-financial, it will return the card data. This can be used for manager cards and private label gift cards that are not processed through Payment Server. The card must be swiped, and the reader must be configured to return the card in unencrypted form.
passthrough	Yes	This action performs a direct pass-through of parameters to the Payment Server server. Only the username, password, and u_action parameters are required. This method will simply proxy requests to the Payment Server and can be used for transaction modifications (e.g. voids) or for pulling reports from the Payment Server. Do not use this request if cardholder data needs to be retrieved for the transaction, it will not be returned. Sensitive cardholder data is explicitly prohibited to be sent as pass-through. Users may wish to send their requests directly to Payment Server rather than using this feature.
cancel	Yes	Will attempt to cancel an outstanding txnrequest or txnstart. Requires username, password and u_id fields which must match the pending request. If the transaction cannot be canceled, such as if it is ineligible (such as when waiting on a response from the Payment Server), or the device doesn't support canceling the outstanding request, u_errorcode will return PENDING_TRAN.
status	Yes	Requests the current status of a pending TXNREQUEST or txnstart/txnfinish. Requires username, password and u_id fields which must match the pending request. This will provide a textual verbiage response suitable for clerk display. Will normally return a u_errorcode of SUCCESS when either the transaction is still in-progress or UniTerm is in a cleanup phase after a transaction. Will return a value of UID_NOT_FOUND if the request is no longer being processed.
deviceupload	Yes	Requests uploading the given file to the device at the requested device path. Not all devices support the concept of file uploads. Required parameters are u_b64data, u_filename, and may return u_needreboot=yes to indicate that a device reboot is required for the change to take effect (an automatic reboot will not occur so that additional files that may also require reboot can be uploaded). The u_filename should contain a properly formatted path for the device in use. Some devices like Ingenico RBA just want the filename with no path component.
deviceprint	Yes	Requests printing the given text to a built-in receipt printer if available. The text provided must be formatted properly for the

		given device. Only plain text receipts are supported. Receipt data is provided via the `u_text` parameter.
`devicereboot`	Yes	Requests the device to reboot. Typically this should only be done when an `u_needreboot=yes` parameter is returned in response to a file upload request.
`reqsignature`	Yes	Requests the device to prompt for signature capture for non-financial requests. This should only be used outside of a transaction flow (e.g `txnrequest` or `txnstart`/`txnfinish`). For financial transactions, a signature will be automatically captured and stored within Payment Server so there is no need to call this function. The signature data will be directly returned to the caller in the `u_signature` response parameter. Not supported in GUI mode
`reqconfirm`	Yes	Requests the user confirm a message presented on the screen. The message is supplied via the `u_message` request parameter, and the response is recorded in the `u_confirmed` response parameter. Not supported in GUI mode
`reqinput`	Yes	Requests the user enter the input type specified via `u_inputtype`. The response data is returned in the `u_input` response parameter. Supported in both GUI and device mode
`devicetypes`	No	Will return a comma separated list of device types supported by the UniTerm module. No authentication required. 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|HID|BLUETOOTH|IP` • `functionality` - pipe separated list of functionality supported by the device. Some devices are families and may list features available in the family but not necessarily the device being used: e.g. `EMV|MAC|E2E|KEY|SIGNATURE|UPLOAD|PRINT|REBOOT|CONFIRM|FREEFORM|POLEDISPLAY` (`FreeForm` is used by `reqinput`)
`seriallist`	No	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	No	List all 'bluetooth' devices that have been paired with the machine UniTerm is running on. The device may or may not be present. Currently only supported on Android and MacOS. 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`
blelist	No	List all BlueTooth LE known devices. If the `u_blelist=scan` flag is set, it will list only currently present devices. When requesting a full scan, an `timeout` key/value pair may be sent, if not sent, it defaults to 30 seconds. Currently only supported on iOS and MacOS. Headers for `u_blelist=scan`: • `devicetype` - Device type as may be passed in `u_devicetype`. • `name` - Textual name as advertised by device. • `identifier` - Unique identifier of device to be passed as part of `u_device=BLE:` Headers when NOT using `u_blelist=scan`: • `devicetype` - Device type as may be passed in `u_devicetype`. • `name` - Textual name as advertised by device. • `service` - Common service exposed by devices of this type, to be passed as part of `u_device=BLESRV:`.
mfilist	No	List all supported Made For iOS paired devices. Headers: • `devicetype` - The devicetype (device internal name) associated with the HID entry • `name` - Textual name as advertised by device. • `protocol` - Protocol advertised by device. • `serialnum` - Serial number of device.
hidlist	No	List all supported HID (possibly USB-HID) devices that are currently attached to the machine UniTerm is running on. Headers: • `devicetype` - The devicetype (device internal name) associated with the HID entry

		• `manufacturer` - The manufacturer as advertised by the device. • `model` - The pretty name for the device type as it is known to UniTerm • `product` - The product as advertised by the device. • `serialnum` - The serial number as advertised by the device. • `path` - System-specific path. May need to be used on Android integrations.
`shutdown`	Yes	Terminates execution of the UniTerm process. This should be called when the application software terminates.
`modifyconfig`	Yes	Allows a select number of `ini` configuration settings to be set via the API. In order to activate the ability to use this feature, an integrator must enable the shared secret in the configuration and the connection must come from the local machine.
`version`	No	Requests the current version of UniTerm. The version information is output in human-readable format in the verbiage response field. The version information also contains the build number.
`deviceinfo`	Yes	Requests the information about the connected device. Returns response parameters of: `serialnum`, `device_manuf`, `device_model`, `device_app`, `device_appver`, `device_kernver`
`standin`	Yes	Perform an operation as specified by the `u_standin` key. For more information on stand in processing in general, see Section 2.4. Please reference the `u_standin` values in Section 5.2.2.
`tab`	Yes	Perform an operation as specified by the `u_tab` key. For more information on ChipTab processing in general, see Section 2.5. Please reference the `u_tab` values in Section 5.2.3.
`expirecached`	Yes	Expire all cached merchant-related data both within the client and host sides to ensure full load data is pulled during the next transaction or `deviceload`.

5.2.2 Stand In Actions (`u_standin`)

Note: Store and Forward, or Stand-in processing requires additional server-side licensing. If licensing the Monetra Payment Server directly, and not using a gateway powered by Monetra such as Transafe, there is a Store and Forward license available which allows all registered UniTerm instances to perform stand-in operations. Otherwise, stand-in processing will consume CardShield device licenses per UniTerm instance.

The table below contains the descriptions of each possible Stand In action.

`u_standin`	description
`update`	Update metadata on file associated with a transaction. These fields can be updated:

	• amount • cashbackamount • clerkid • comments • custref • examount • ordernum • ptrannum • stationid • tax
send	Initiate sending stored transactions to the Payment Server. Transactions are sent on a configurable timer and can be triggered to start sooner with this sub action. Transactions will attempt to be sent but transactions may remain in a stored state if there are connectivity issues. Since this triggers the timer it applies to stored transactions of all users on the system. You cannot selectivity send transactions. This returns immediately and starts sending transactions as a background task within UniTerm. It does not return any indication about status of or number of transactions.
void	Remove a transaction that has not yet been sent online. Must pass the `ttid` returned from UniTerm.
list	List all stored transactions for the user returning a CSV result. These keys can be used to alter the query results (these should align with query parameters as accepted by the Payment Server): • ttid • bdate • edate • timestamp • type - (action) • amount • examount • tax • card - (cardtype) • pclevel • account - (last 4) • expdate • cardholdername • ordernum • custref • ptrannum • clerkid • stationid • comments

	The resulting CSV will contain these headers:
	`ttid,proc,type,entrymode,card,pclevel,account,amount,`
	`cardholdername,clerkid,comments,custref,expdate,`
	`examount,cashback,ordernum,ptrannum,stationid,tax,`
	`timestamp`

`totals`	Number of stored transactions and total stored amount for the requesting user. Will return these result keys: • `num_trans` - Number of transactions on file that have not yet been forwarded • `total_amount` - Total dollar amount (sum) of transactions on file, not yet forwarded
`listresp`	A CSV report with basic information about stored transaction that have been successfully sent online. The TTID returned by UniTerm when stored will appear in the `u_ttid` column and the Payment Server's TTID will appear in the `ttid` column. Follow up transactions, such as reversal, can continue to use the UniTerm TTID as long as the response is still on file. It is highly recommended that the POS replace the UniTerm TTID with the Payment Server TTID internally as soon as possible. The resulting CSV will contain these headers: `u_ttid,ttid,timestamp,code,verbiage`
`getresp`	Get the Payment Server response for a stored transaction that was successfully sent online. This will return the full key value pair result returned by the Payment Server. May pass the `ttid` as either the UniTerm-returned ttid or the Payment Server ttid as retrieved from `u_standin=listresp`
`delresp`	Purge the Payment Server response for a stored transaction that was successfully sent online. May pass the `ttid` as either the UniTerm-returned ttid or the Payment Server ttid as retrieved from `u_standin=listresp`

5.2.3 ChipTab Actions (`u_tab`)

Note: ChipTab requires additional server-side licensing. The same Store and Forward license as required for Stand-in processing is also required for ChipTab support. If already performing stand-in, then no additional licensing is necessary.

The table below contains the descriptions of each possible ChipTab action.

u_tab	description
`open`	Open a tab and capture customer card data. A TTID will be returned on success which is used to reference the tab for follow up sub actions. The TTID will be prefixed by 'U' to indicate the TTID is associated with UniTerm.

	This takes the same parameters as a `txnstart` would take such as `u_device`, `u_devicetype` and so on. Note: An amount is NOT specified on open.
`close`	Close an existing tab. If a device (via `u_device` and `u_devicetype` is not specified this works the same as force closing a tab. When using a device the customer will be prompted to accept the amount and sign if necessary. Must specify the `ttid` returned during `open`. If an amount is not already associated with a tab (by updating the tab) it is required to pass the amount. An amount can be passed even if an amount is on file and will be used instead of what is on file. `u_up_amount` can be used when closing a tab.
`force_close`	The transaction is sent directly to the Payment Server without customer interaction. Useful when a customer leaves without explicitly closing their tab. See `close` for details about required and optional parameters.
`update`	Update metadata on file associated with a transaction by specifying the `ttid` associated with the tab. These fields can be updated: • `amount` • `cashbackamount` • `clerkid` • `comments` • `custref` • `examount` • `ordernum` • `ptrannum` • `stationid` • `tax` Additionally, tabs can have the stored amount increased or decreased using the `u_up_amount` key. Positive numbers will increase and negative will decrease. This is useful if the POS wants UniTerm to track the tab amount instead of tracking itself. For example, the current amount is $15.00. Sending `u_up_amount=2.00` will change the current amount that is stored to $17.00. Next, sending `u_up_amount=-3.00` will reduce the amount to $14.00. Note: `u_up_amount` is different than `amount`. Amount is the total and sending this key will replace the current value. `u_up_amount` modifies the current amount.
`void`	Deletes a tab. The tab is closed but never sent online for authorization. Must specify the `ttid` from the `open` request.
`list`	List all tabs for the user returning a CSV result. These keys can be used to alter the query results (these should align with query parameters as accepted by the Payment Server):

- `ttid`
- `bdate`
- `edate`
- `timestamp`
- `type` - (action)
- `amount`
- `examount`
- `tax`
- `card` - (cardtype)
- `pclevel`
- `account` - (last 4)
- `expdate`
- `cardholdername`
- `ordernum`
- `custref`
- `ptrannum`
- `clerkid`
- `stationid`
- `comments`

The resulting CSV will contain these headers:
`ttid,proc,type,entrymode,card,pclevel,account,amount,`
`cardholdername,clerkid,comments,custref,expdate,`
`examount,cashback,ordernum,ptrannum,stationid,tax,`
`timestamp`

`totals`	Total number of open tabs and total tracked amount of across all tabs. The total amount is only useful is using UniTerm to track the total for each tab by utilizing `u_up_amount`. If the POS is tracking amount itself and only sending the amount when closing the tab the aggregate total amount will most likely be $0.00 and not accurately reflect the outstanding total amount. Will return these result keys: • `num_trans` - Number of transactions on file that have not yet been forwarded • `total_amount` - Total dollar amount (sum) of transactions on file, not yet forwarded

5.2.4 Interchange/Rate Qualification Requirements

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

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

Key	Required When	Description
nsf	Card Present	Required value: yes Approve transactions even if there are insufficient funds (NSF). Will result in a partial approval if there are insufficient funds, an authamount response parameter will be returned for partial authorizations. When a partial authorization is received, a merchant should request another payment method for the remaining funds. It is acceptable to request a reversal if no additional payment methods are possible. All card brands are requiring merchants support partial authorizations, merchants can be subjected to fines. Some issuers may return partial approvals even without this flag because of the requirements in place.
laneid	Card Present	The lane ID is a Mastercard requirement that each register or lane at a merchant's location be uniquely identified. This should be a numeric value no more than 8 digits in length, however some processors cannot support more than 2 or 4 digits. It should be sent on all transactions as it is not possible to know the card type prior to a request to UniTerm.
ordernum	Restaurant, Card Not Present, Level 2	A merchant order number is required on all card not present (mail order and ecommerce) transactions, Restaurant, and all Level2 cards such as purchase or corporate cards. Since it is not possible to know the card type prior to a request to UniTerm an Order Number should be sent on all transactions. An order number is normally alpha-numeric up to 25 characters, however for restaurant a 6 digit order number should be used
examount	Restaurant	Extra Amount - Tip Amount. If a tip amount is known at the time of sale it must be populated, if using UniTerm's built-in tip-prompting, this field should not be passed as UniTerm will populate it.
zip	Key Entered	All key entered transactions should include at least a billing zipcode for best rate qualification. Often this is best accomplished by passing a u_flags of AVS, however there may be cases where the billing zipcode is already on file and will be presented separately.
healthcare	HSA, FSA, HRA-qualified transactions	If attempting to perform a health care-qualified transaction, this must be indicated. Should also send rxamount, dentalamount, transitamount, clinicamount, visionamount, or otheramount if available.

5.2.5 Shared Secret/HMAC handling

A shared secret is used to authenticate commands that do not authenticate using a Payment Server user. It is required to use a shared secret for remote connections (`localonly=no`) if unauthenticated requests need to be used. Authenticated requests such as `u_action=txnrequest` do not use the shared secret, even if originating outside the local machine.

The shared secret is used as the key to generate a transaction-specific `HMAC-SHA256` parameter sent with the requests to UniTerm. A u_id must be sent with transactions that require using the shared secret.

The following key value pairs are required when using a shared secret, their values are concatenated together for the `HMAC` calculation:

- `u_action`: UniTerm action being performed
- `u_id`: Unique identifier for this transaction
- `u_req_sequence`: A sequence number that should not repeat
- `u_req_timestamp`: Unix Timestamp. Must be within 1 minute of the UniTerm server's time.

The `u_req_hmacsha256` parameter is sent with the calculated `HMAC-SHA256` value from the above calculation.

For example, assuming a transaction with these fields:

```
u_action=version
u_id=1
u_req_sequence=1202
u_req_timestamp=1509043043
```

would use `version112021509043043` as the input to the `HMAC-SHA256`, and if the shared secret is `secret` the HMAC would calculate to:

```
6E6C0CDF15E43DACDFB7DC24598EFF95FE3897AF0FF6917666DC454B0437FE0D
```

For more information on HMAC calculation, please reference Appendix B.

Note: For `u_action=modifyconfig` requests, the `sharedsecret_mc` configuration value will be used instead.

5.3 UniTerm Response Parameters

The UniTerm module will return all standard response tags from the Payment 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) = Payment Server returns rcpt_emv_cvm which can have a value of "sig" saying signature is required. The u_need_signature means that a signature is required and it should be printed/obtained from the paper receipt. If an EMV requires a signature and one was not captured electronically, this flag indicates it should be obtained via a paper receipt.
u_errorcode	See section below.
u_cancelable	u_action=status only. Yes or No. Indicates if the current transaction state will allow a cancel to be sent. This is useful for showing and hiding a cancel button within an integration's GUI.
trackdata	u_action is txnrequest, txnstart, or cardrequest only. Also requires the ini configuration of unsupportedcard=yes. Will only be returned for non-financial cards that are returned unencrypted from the reader. Facilitates the use of manager cards as well as in-store private label gift systems that do not flow through the PAyment Server. The u_errorcode returned with the response will always be NONFINANCIAL. Support during a txnrequest/txnstart is tailored to the use of private label gift cards and will only be returned when the cardholder selects GIFT from the payment type selection screen.
serialnum	u_action=deviceinfo only. Serial Number of connected device.
device_manuf	u_action=deviceinfo only. Device Manufacturer.
device_model	u_action=deviceinfo only. Device Model Name.
device_app	u_action=deviceinfo only. Application running on connected device.
device_appver	u_action=deviceinfo only. Version of the application running on the connected device.
device_kernver	u_action=deviceinfo only. EMV Level2 kernel version of the connected device.
device_encryption	u_action=deviceinfo only. Encryption type used by device, if any.
u_needreboot	u_action=deviceupload only. Will return "yes" when a device reboot is needed for the action performed to take effect.

u_signature	`u_action=reqsignature` only. Will return base64-encoded TIFF data representing the signature captured by the device.
u_confirmed	`u_action=reqconfirm` only. Will return "yes" or "no" depending on if the user confirmed the dialog.
u_input	`u_action=reqinput` only. Will return the user-entered data.
u_tip	`u_action` is `txnrequest` or `txnfinish`. Will return the tip amount chosen by the cardholder.
u_cashback	`u_action` is `txnrequest` or `txnfinish`. Will return the Cash Back amount chosen by the cardholder.
u_wasfood	`u_action` is `txnrequest` or `txnfinish`. Will return a boolean value indicating if the cardholder selected EBT Food Stamps (SNAP).
u_status	`u_action` is `status`. Machine readable status code. See UniTerm Status Codes.
u_represent	`u_action` is `status`. Machine readable reason code for why card representment is being requested. See UniTerm Representment Codes.
u_flowflags	`u_action` is `txnrequest`, `txnstart`, or `txnfinish`. Pipe-delimited flags to provide integrators more insight into possible actions UniTerm has taken. • `SIGCAPTURED` - Signature was captured • `REVERSED` - The transaction was approved but then reversed. This is typically due to the customer canceling the transaction, or the card declined the transaction.
u_standin	If transaction was the result of a Stand-in, this response parameter will be returned with a value of `yes`.
u_cardclass	The card class associated with a transaction. In most cases this should be ignored as a more specific card type is returned by the `cardtype` response parameter, however with `EBT` transactions the card class is used to indicate if a Cash Benefits or Food Stamps (SNAP) transaction is being performed. Possible responses: • `CREDIT` - Credit Card • `DEBIT` - Debit Card • `GIFT` - Gift Card • `EBTFS` - `EBT` Food Stamps (SNAP) • `EBTCB` - `EBT` Cash Benefits
u_rcpt_*cust_*	Series of customer-specific receipt sections. See Appendix C for more information.

| u_rcpt_*merch_* | Series of merchant-specific receipt sections. See Appendix C for more information. |

5.4 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 Payment Server were successful. It does not mean the transaction was approved.

u_errorcode	definition
FAILURE	Generic Failure
SUCCESS	UniTerm successfully processed the transaction. This does not, however, mean the transaction was approved.
MISSING_PARAM	A required parameter was missing.
INVALID_PARAM	A specified parameter was invalid
PENDING_TRAN	pending transaction already in progress
UID_NOT_FOUND	A `u_id` was specified on a status or cancel request and no such `u_id` is actively being processed.
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.
INVALID_USE	Typically means parameters specified should not have been specified together.
INTERNAL_ERROR	UniTerm has encountered an internal problem.
PERMISSION_ERROR	The user account within the Payment Server was misconfigured.
MONETRA_ERROR	There was an error communicating with the Payment Server.
MONETRA_PROC_ERROR	The processor returned an error.
MONETRA_DATA_MISMATCH	Data received from the Payment Server is in wrong format or expected data is missing.
MONETRA_CONNFAIL	Connection to the Payment Server failed.
MONETRA_TIMEOUT	Payment Server response timed out.
DEVICE_INUSE	The device specified is being used by another transaction.
DEVICE_ERROR	There was an error communicating with the card entry device.
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.
`CANCELED`	User canceled request
`TIMEOUT`	Timeout waiting for user input or device response.
`TRY_AGAIN`	Transaction cannot be completed at this time, retry.
`DBFAILURE`	Database has encountered an error.
`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_PARSE`	The `uniterm.ini` format is invalid.
`INI_CANNOT_WRITE`	The `uniterm.ini` is not writable by the UniTerm process.
`INI_INVALID_PARAM`	The `uniterm.ini` has an invalid configuration parameter.

5.5 UniTerm Status Codes

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

`u_status`	definition
`WORKING`	Generic status say UniTerm is processing.
`CARD_IST`	Waiting for card presentation (Insert, Swipe, or Tap). With an EMV card, the card could be inserted and waiting for customer to choose language.
`CARD_IT`	Waiting for card presentation (Insert or Tap). With an EMV card, the card could be inserted and waiting for customer to choose language.
`CARD_IS`	Waiting for card presentation (Insert or Swipe). With an EMV card, the card could be inserted and waiting for customer to choose language.
`CARD_ST`	Waiting for card presentation (Swipe or Tap), non-EMV.
`CARD_I`	Waiting for card presentation (Insert only). With an EMV card, the card could be inserted and waiting for customer to choose language.
`CARD_S`	Waiting for card presentation (Swipe only).
`CARD_T`	Waiting for card presentation (Tap only).
`EMVFLOW`	EMV processing. Typically waiting for customer to enter their PIN.
`KEYED`	Waiting for keyed account and related (zip, cvv).
`PIN`	Waiting for PIN entry. This may not be presented for EMV cards as the device may handle this internally.

CARDCLASS	Waiting for card class selection (Debit, EBT, Gift prompting)
TIP	Waiting for tip entry.
CASHBACK	Waiting for cash back entry.
SIGNATURE	Waiting for signature.
EMVCOMPLETE	EMV processing is completing.
CONFIRM	Waiting for user confirmation.
CONFIRMAMOUNT	Waiting for user to confirm the transaction amount.
CONFIRMFOOD	Waiting for user to confirm the food amount.
CONFIRMRETRY	Waiting for user to confirm to retry the transaction after a failure.
INPUT	Waiting on customer input (e.g. phone number).
MONETRA	Communication with the Payment Server. Typically waiting on a response from the Payment Server.
REMOVECARD	Waiting for card removal.
REVERSAL	Reversing transaction (typically due to card decline).
DEVICEPRINT	Printing receipt on device.
DEVICEOPEN	Waiting for device to open.
DEVICECLOSE	Waiting for device to close.
DEVICEREBOOT	Rebooting device.
CANCEL	Canceling transaction or operation.

5.6 UniTerm Representment Codes

Representment codes returned via the u_represent response parameter. This response parameter gives a reason that card representment is being requested.

u_represent	definition
BAD_INSERT	Chip Read Failure. Could be bad card, card inserted backwards or upside-down, or customer removed the card too early.
BAD_SWIPE	Failure reading Magnetic card data. Possibly damaged card or customer swiped too quickly.
BAD_TAP	Failure reading NFC card data. Possibly damaged card or customer did not hold the card close enough to reader for a long enough period of time.
CHIP_CARD_SWIPED	Customer attempted to swipe a chip card instead of inserting it.
OVER_TRAN_LIMIT_TAP	Transaction exceeds maximum dollar amount, tap is not allowed.
RETRY	Retry for some other generic failure reason.
UNSUPPORTED_CARD	Card presented is not usable. Try a different entry method or card.

5.7 Tip Prompting

UniTerm supports prompting the cardholder for a tip amount at the time of payment when the Payment Server merchant account configuration `merch_tippercent` setting is configured and the `NOTIP u_flags` parameter is NOT provided.

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

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

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

5.8 Cash Back Prompting

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

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

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

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

5.9 EBT Processing

UniTerm supports prompting if the card presented is an EBT card. If the `u_foodamount` parameter is populated with a non-zero dollar amount indicating the amount of the transaction

---

that applies to qualified food purchases (or for `txnstart` with a value of `maybe`), then the customer will also be prompted if they would like to use Food Stamps (SNAP) or Cash Benefits to complete the transaction. When the cardholder makes the selection, UniTerm will internally rewrite the `action=sale` request parameter to `action=ebtfssale` or `action=ebtcbsale` as appropriate.

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

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

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

5.10 QuickChip

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

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

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

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

5.11 Pay at the Table

UniTerm provides all the tools needed for an integrator to support Pay at the Table. Though the method an integrator might choose can vary from system to system, we have outlined a suggested flow that should work for most environments when integrators choose to use a

---

payment device specifically meant for pay at the table without necessitating the need for the use of an additional device such as a tablet. Those integrators that choose to use a tablet for pay at the table will not use the information contained in this section as the flow, in relation to UniTerm, would more closely resemble the standard Pay at the Counter flow.

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

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

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

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

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

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

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

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

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

- Start the flow over for more tickets.

5.12 Parking: Card-In/Card-Out Entry and Exit Gates

Using a Credit Card as the identity token rather than a printed ticket is a common practice at some parking garages. Due to security requirements, it is highly desirable to keep the cardholder data itself outside of the parking system and use unique identifiers (tokens) from the Payment System for this purpose. Due to the complexities of the EMV transactional flow it may not be obvious how to accomplish this without requiring multiple card presentments.

We have outlined our recommended best practices for implementing this functionality with UniTerm.

1. A customer arrives at the entry gate.

   a. A `txnrequest` is performed with these key/value pairs:

   ```
   u_action=txnrequest
   action=admin
   admin=recurringadd
   type=store
   matching_token=yes
   ```

   Note: Using `matching_token` in the request is a required part of this process to ensure a new token is not created if the card number is already in the system.

   b. Customer will see "administrative" on the device which indicates it is not a financial transaction they are performing at that moment

   c. A `token` response parameter is returned by UniTerm which the parking system will keep and use as the unique identifier.

2. Customer leaves parking facility and is at the exit gate.

   a. A `txnstart` (not `txnrequest`!), is performed with these key/value pairs:

   ```
   u_action=txnstart
   action=sale
   u_id=$uniqueid
   tokenize=yes
   matching_token=yes
   ```

   b. A `token` will be returned in the response. This token will be used to match against the parking system's entry gate list to determine the amount owed. The device will automatically show "Please Wait" until the next step.

   c. Send `txnfinish` using the `u_id` passed during `txnstart` and the determined amount from the prior step

   ```
   u_action=txnfinish
   u_id=$uniqueid_from_txnstart
   amount=$parkingfee
   ```

   The customer's card will be charged as card present and the token will not actually be used for the transaction itself as the card was already presented to the exit gate so it has access to the full EMV data necessary.

   d. For "house-keeping" purposes, it may be desirable to perform a `recurringdel` request to clean up the card on file when no longer needed.

5.12.1 Considerations

Some issuers will assign the same card number to joint accounts. The system recording the token entries needs to account for seeing the same token enter multiple times. For example, If

it's an airport and the cardholders take two cars because it's a large family you could end up with two enter events for the same card.

Another situation to consider is if the customer's card is lost, stolen or otherwise compromised while they're away, again think airport parking. They may not have or be able to use the card they entered with when they leave.

5.13 Signature Capture

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

Note: Some devices, especially mobile devices, or web browsers may not natively support displaying the `TIFF` image format. Recent versions of the Payment Server support conversion to `PNG` when requesting a `getimages` request by passing `image_format=PNG` with the request. Otherwise if a different image format is desired, or using an older version of the Payment Server, it is recommended to use an open source library such as `ImageMagick` to assist in the conversion for display purposes.

6 EMV transactions with UniTerm

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

6.1 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.

6.1.1 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.

6.1.2 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).

6.1.3 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.

6.1.4 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.

6.1.5 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.

6.1.6 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.

6.1.7 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.

6.2 Common questions

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

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

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

6.2.2 What industries are certified for EMV?

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

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

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

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

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

7 Storage, Key Management, and Logging

UniTerm v9 introduced a database backend for caching merchant and transaction data for offline processing, as well as logging facilities for monitoring and inspection of transaction processing.

7.1 Database Storage, Security, and Key Management

UniTerm supports data storage to just about any database backend, though the most commonly deployed is SQLite due to its zero-configuration nature. All sensitive data is encrypted prior to hitting the database layer using field-level encryption. Database functionality is facilitated by the `mstdlib` open-source library available at: https://github.com/Monetra/mstdlib. The location of data storage is defined in the database configuration Section 4.2.4.

The UniTerm database serves two purposes. The first purpose is as a caching front-end for all of the loading data necessary to operate a terminal. This includes, but is not limited to, merchant metadata (name, address), EMV terminal parameters (AIDs, CAPKs), and card BIN tables. The second purpose is for Stand-in authorizations and ChipTab support, where sensitive cardholder data is stored until authorization, as well as tracking of metadata after authorization.

Data is cached to disk from the Payment Server to limit necessary round-trips which may add latency, especially if UniTerm is frequently stopped and restarted (as is the case on Mobile devices to conserve power), as well as facilitating offline terminal loading. Cache refresh times vary, but in general merchant-specific settings are refreshed every 24hrs and Payment-Server-specific parameters such as large BIN tables may not be updated more than once per week. UniTerm will forcibly re-cache merchant data if a `u_action=deviceload` request is sent.

For Stand-in and ChipTab support, sensitive cardholder data is stored encrypted using `AES-256-CBC` in the `stored_trans` table per key/value pair. The `AES` key used is an ephemeral key generated for that specific transaction, and is stored in the same `stored_trans` table in the `e_id` key mapping which itself is encrypted using a merchant-specific `RSA` public key retrieved from the Payment Server (but not private key which is generated by and private to the Payment Server) utilizing its validated CardShield subsystem. Once the transaction is encrypted, all in-memory traces are wiped and UniTerm can no longer access the sensitive cardholder data which can now only be decrypted by the Remote Payment Server instance holding the `RSA` private key. The merchant-specific `RSA` public keys are stored in the `stored_keys` table and are rotated automatically every 10-12 days. After authorization, all transaction request data is purged from the database. Data after authorization may be stored in the Payment Server and is up to the administrator of the remote Payment Server system to configure retention periods as per customer requirements.

Note: Even when using an encrypting reader, the encrypted output will be re-encrypted using the above method before being stored to the database.

For offline operation, requests must still be authenticated, therefore the last successfully validated password for the merchant is recorded to the `user` table using the secure hashing mechanism `PBKDF2-SHA256` and the number of rounds as specified in the `password_iterations` configuration parameter (default is 10000). Once the password has been validated the first time, it is then cached in memory in an `AES-256-CBC` encrypted hash table to reduce the CPU load required in re-computing the hash for verification while still staying secure from common attack vectors such as memory scrapers.

Note: There is no ability to control this card storage or key rotation behavior, it is always implemented as per above. However if a forced key rotation is desired sooner than the normal 10-12 day period, then the merchant can delete the entire UniTerm database which will force a new key to be generated for the UniTerm instance within the Payment Server.

7.2 Logging

The UniTerm log is a multi-purpose log format. It provides insight into the operations of UniTerm as well as a detailed audit trail of every operation performed by an end user, including, but not limited to, IP address, request types along with request and response parameters, and duration of connection. UniTerm's logging does not exist to meet any PCI requirements since UniTerm does not provide any sort of access to cardholder data that would require authentication and is generally not relevant to PCI, but rather for debuggability and insight into system function.

The UniTerm log will never log sensitive merchant or cardholder data, any data that may be considered sensitive is completely masked in the logs.

UniTerm facilitates centralized logging via remote syslog facilities, either by using a system-provided syslog API, or by directly supporting Syslog over TCP. Please see Section 4.2.5 for more information on available logging configuration parameters.

PCI DSS mandates that logging of all modifications to system level objects is done on systems containing payment applications. It is required that merchants deploy a logging facility such as file integrity monitoring on such payment systems to comply with this requirement. UniTerm does not provide such a facility directly, and is an external requirement which must be implemented by the merchant.

Note: There is no ability to configure UniTerm to log in an insecure manner. The default logging levels are recommended and validated to be PCI compliant.

7.3 External Data Storage

If the merchant backs up the UniTerm database or otherwise stores sensitive data outside of the controls put in place by UniTerm or the Payment Server, it is the merchant's responsibility to do so in a PCI DSS compliant manner for the customer-defined retention period.

Note: Merchants must not back up any swap files as they may contain sensitive cardholder data meant to only exist in volatile memory.

---

8 UniTerm Protocol Examples

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

8.1 EMV Transaction [device load]

8.1.1 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

8.1.2 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

PARAMETER	VALUE
termcaps_loa	E0B8C8
termtype	22
termtype_desired	21
u_errorcode	SUCCESS
verbiage	Device loaded

8.2 EMV Transaction [Interac]

8.2.1 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

8.2.2 UniTerm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXXXXXXX2145
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

8.3 Transaction Request with EBT Food Stamp optional

8.3.1 UniTerm Request Data

PARAMETER	VALUE
action	sale
amount	10.00
u_foodamount	5.00
nsf	yes
ordernum	899065992
password	test123
u_action	txnrequest
u_device	HID
u_devicetype	ingenico_rba

u_id	899065992
username	transarmor_isc250:sub

8.3.2 GUI output

8.3.3 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

9 UniTerm Test Application

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

10 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 E 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

11 UniTerm Hardware Devices (Point of Interaction Devices)

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

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

11.1 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 the Payment Server 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					
Ingenico `RBA` family (`iPP320`,	RBA	`ingenico_rba`	USA	CardShield, First Data `TransArmor` RSA, `OnGuard`	x

---

Model	Device S/W	u_devicetype	Notes	Encryption	EMV
iSC Touch 250, etc)					
Ingenico UPP/Tetra family (Lane/3000, Lane/5000, Lane/7000, Lane/8000, Link/2500, Move/3500, Move/5000, etc)	UPP	ingenico_upp	USA	CardShield (Generic Triple-DES)	x
iUP250 TCPX	CPX	ingenico_tcpx	Canada	NONE	x
Verifone					
vx805	XPI	verifone_vx	USA	NONE	x
BBPos/Anywhere Commerce					
Chipper 2X BT/Walker C2X BT	CROS	bbpos_cros	USA	CardShield	x
IDTech					
Augusta	NGA	idtech_augusta	USA	CardShield	x
Spectrum Pro	NGA	idtech_spectrum _pro	USA	CardShield	x

11.1.1 Ingenico RBA and UPP information

11.1.1.1 RBA firmware versions and devices

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

- iCMP
- iSMP companion
- iSMP4
- iWL250
- iPP320 - PCI PTS v3.x+ version
- iPP350 - PCI PTS v3.x+ version
- iSC Touch 250
- iSC Touch 350
- iSC Touch 480
- iUN - iUP250, iUR250, iUC150
- iUC285

The minimum version of the RBA software load supported is 14.0.6, though newer is more preferred as there may be bug fixes or additional features available on newer releases. There are two recommended versions depending on the EMV Certification being used, please cross-reference the kernel number of the certification in use with the kernel associated with the RBA version.

A list of recommended/supported versions past and present are below:

- `23.0.38 - 5.26 kernel - UniTerm 9.3.0+ (current)`
- `23.0.38 - 5.26 kernel - UniTerm 9.3.0+ (current) - "Unattended" series`
- `23.0.24 - 5.26 kernel - UniTerm 9.3.0+ - no longer recommended due to contactless EMV issues`
- `23.0.28 - 5.26 kernel - UniTerm 9.3.0+ - "Unattended" series - no longer recommended due to contactless EMV issues`
- `23.0.12 - 5.26 kernel - UniTerm 8.4.5+`
- `23.0.16 - 5.26 kernel - UniTerm 8.4.5+ - "Unattended" series`
- `21.5.6 - 4.67 kernel - UniTerm 8.3+ (current)`
- `23.0.2 - 5.26 kernel - UniTerm 8.4.5+`
- `23.0.4 - 5.26 kernel - UniTerm 8.4.5+ - "Unattended" series`
- `21.7.2 - 5.26 kernel - UniTerm 8.4+`
- `21.0.2 - 5.26 kernel - UniTerm 8.4+`
- `19.0.8 - 4.67 kernel - UniTerm 8.2+`
- `16.0.2 - 4.67 kernel - UniTerm 8+ - no longer recommended due to contactless MSD issues`
- `15.0.6 - 4.67 kernel - UniTerm 8+`
- `14.0.6 - 4.67 kernel - UniTerm 8+`

> Note: Use of any version of RBA not explicitly listed as recommended is at the users own risk and no guaranteed support can be provided. Some features might require newer versions of RBA so it is always recommended to run the latest recommended release.

11.1.1.2 UPP firmware versions and devices

The UPP family includes all Ingenico `Tetra` devices that can run the UPP (Universal Payment Platform) software. This includes, but is not limited to:

- `Lane/3000,5000,7000,8000`
- `Link/2500`
- `Move/3000,5000`

A list of recommended/supported versions past and present are below:

- `5.08.05 - 8.38 kernel - UniTerm 9.4.0+ (current)`
- `5.07.00 - 8.38 kernel - UniTerm 9.1.0+`

> Note: Use of any version of UPP not explicitly listed as recommended is at the users own risk and no guaranteed support can be provided. Some features might require newer versions of UPP so it is always recommended to run the latest recommended release.

11.1.1.3 Communication Methods

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

- `USB-HID` - No drivers are necessary, select HID as the device connectivity in UniTerm. Not available on `UPP` devices.
- `USB<>Serial Conv` - Requires `Telium` or `Jungo` drivers on Windows, will show up as a virtual COM port and be used as a Serial device in UniTerm (Linux and MacOSX do not need drivers, they will show up as serial ports).
- `Serial` - Settings: `115200 8N1` - No flow control
- `Bluetooth` - Android native, on Windows it should show up as a virtual COM port and be usable as a Serial device in UniTerm. Not currently available on `UPP` devices due to introduction of `PCL` layer, expected to be resolved as of `UPP 6.0.3`.
- `MFI` - iOS native, Made for iOS, via BlueTooth. Not currently available on `UPP` devices due to introduction of `PCL` layer, expected to be resolved as of `UPP 6.0.3`.
- `Ethernet` - Both IP Server (UniTerm -> Device) and IP Client (Device -> UniTerm) connectivity supported. IP Client mode also supports SSL/TLS.

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

11.1.1.4 Device configuration

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

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

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

For `UPP` the code is the same, however you can simply enter it at the idle screen rather than rebooting the device.

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

11.1.1.4.1 `RBA` Contactless Support

Some devices such as the `iPP320` have contactless as an optional module. It is possible when such a device is shipped to you, the optional module is installed but not activated. If you are certain the device has the necessary hardware for contactless, but contactless is not working,

---

you may need to activate it. For the `iPP320` this can be confirmed by observing the existence of a contactless chip behind the rear door of the device. Contactless must not be enabled if the device does not have the proper hardware.

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

11.1.1.5 `RBA` 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 higher version of the devices. Older hardware revisions comply with `PCI PTS v2` and should not be used for new deployments as you may experience issues due to limitations in the hardware.

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

11.1.1.6 Forms and Images

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

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

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

The forms used and their requirements are listed below.

Forms and Images used by UniTerm:

- `UTAD.K3Z` - The form displayed when the device is idle, also known as the "ADs" screen. This form may be customized to present an image or a series of rotating images, but must

---

not contain buttons. The default form loaded contains a single image, `UTAD.PNG` (or `UTAD.BMP` on `iUP250/iUC285` devices). It is recommended that the images created be specific to the device for best appearance even though the device will scale the image if too small or large.

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

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

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

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

- `AMTV.K3Z` - Form used to display confirmation prompts, both for arbitrary prompts and amount confirmation. This is a stock form, any replacements should adhere to the capabilities of the stock form.

- `UTASEL.K3Z` - Form used to display tip and Cash Back prompts, based on the `cashba.K3Z` stock form. Button IDs must be:
  - `O` - Other
  - `N` - No (won't be shown on smaller devices)
  - `A` - Amount 1
  - `B` - Amount 2
  - `C` - Amount 3

- Ingenico may internally call additional forms during the EMV payment processing flow. For information on how to customize these screens, integrators should contact Ingenico.

11.1.1.7 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 the Payment Server. When configuring the Payment Server 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 the Payment Server which will request the current key to use from First Data's systems. The Payment Server 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.

11.1.1.8 Updating `RBA` or `UPP` firmware with UniTerm

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

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

For convenience, validated firmware versions for `OGZ` upgrades have been made available here: https://download.monetra.com/devices/ingenico.

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

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

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

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

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

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

---

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

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

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

   For `UPP` devices, the Generic Triple-DES (`TDES`) activation file should be used.

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

11.1.1.9 SSL Client (Device -> UniTerm) Mode of operation

A `PGZ` with the client certificate needs to be loaded onto the device. Ingenico refers to the device as the server no matter how it's configured. So care must be taken with the files to ensure they're not mixed up between the device (client) and server (UniTerm). `RBA/UPP` must have a certificate signed by the same CA as the server. `RBA/UPP` does not do hostname validation and uses the CA loaded on the device for verification of the server. It is up to the integrator to generate a certificate for the device to use and the CA for verification and share this with Ingenico who will generate the `PGZ` file for the device.

If using self signed certificates, generate a CA certificate and private key. Generate a private key and certificate signing request for both the client and server. Sign each request with the CA to generate a certificate. Send the client certificate, client private key, and CA certificate to Ingenico. They will package these three files into a `PGZ` that will be loaded onto the device.

To enable SSL mode via the device menus, set:

- `TDA -> Configuration -> Communication -> Select Comm. Type -> Ethernet`
- `TDA -> Configuration -> Communication -> Ethernet Settings -> Connection Method -> Client`
- `TDA -> Configuration -> Communication -> Ethernet Settings -> DHCP` - Set Auto if using `DHCP`, or Static if not
- If using Static: `TDA -> Configuration -> Communication -> Ethernet Settings -> IP Address` - Enter IP address to use
- `TDA -> Configuration -> Communication -> Ethernet Settings -> Host IP Address` - Enter IP address of UniTerm server
- `TDA -> Configuration -> Communication -> Ethernet Settings -> Host IP Port` - Enter Port number UniTerm is configured to listen on

- TDA -> Configuration -> Communication -> Ethernet Settings -> SSL -> Yes

11.1.2 Verifone `VX` `XPI` information

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

11.1.2.1 Communication Methods

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

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

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

11.1.2.2 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.

11.1.3 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.

11.1.3.1 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.

11.1.4 Ingenico `TCPX` information

Ingenico `TCPX` replaces `CPX/uCPX` firmware and is designed to power both attended and unattended Ingenico terminals in Canada and includes support for Interac debit cards.

The recommended firmware version is `20.26A` with kernel `5.38`.

11.1.4.1 Communication Methods

UniTerm supports communicating with `TCPX` 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`

11.1.5 BBPos Chipper 2X BT/Anywhere Commerce Walker C2X BT information

The BBPos Chipper 2X BT device is supported on all platforms and is optimized for Mobile payments. The device does not have a pinpad or display. The minimum firmware version supported is 1.00.03.32 with configuration `MQZZ_mssw_v12`.

11.1.5.1 Communication Methods

UniTerm supports communicating via these communication methods:

- `USB HID` - Linux/MacOS/Windows/Android
- `Bluetooth Classic` - Android/MacOS
- `Bluetooth LE` - iOS/MacOS

11.1.6 IDTech Augusta and Spectrum Pro (`NGA`)

The Augusta device is an MSR + EMV Contact reader connected via USB HID meant to replace the classic MSR-only card readers found on classic POS systems.

The Spectrum Pro device is an MSR Insert + EMV Contact reader connected via USB HID meant for stand-alone kiosks. An optional Pin Pad may be attached to offer a display and PIN entry.

11.2 Obtaining Devices

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

- Is the device supported by UniTerm?
  - Please see the prior section: Section 11.1
- Is the device certified for the Processing Institution being used?
  - Please see the next chapter: Section 12.1
- Does the device come preloaded with both the right software and version?
  - Please see section specific to the chosen device under Section 11.1 for supported software revisions running on the device.
- Is the vendor the device is sourced from able to load the appropriate keys (Pin Debit or Encryption) as required?
  - Please see the next section: Section 11.2.1

11.2.1 Where to source devices with appropriate loads and keys

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

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

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

If Point to Point Encryption is also desired, it is necessary to have your key injection facility also load that key into your device and enable the appropriate encryption support in the

---

software to utilize the key before shipping the device. One notable exception is First Data's RSA `TransArmor` on `Ingenico RBA` devices does not need to be loaded as UniTerm can load the appropriate keys itself. If using CardShield encryption, it is necessary to have first shared a Base Derivation Key with your key injection facility from your instance of the Payment Server, in order for them to be able to load the key into your device and enable the support before shipping. If using TranSafe a list of key injection facilities along with the key identifiers is provided at https://www.transafe.com/.

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

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

12 Certifications and Device Configurations

12.1 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				
Verifone vx805	6.2.0/1C	8.0+	Paymentech Tampa 3.2.0 (Done Nov 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Ingenico RBA family	4.67/1C	8.0+	Paymentech Tampa 3.2.0 (Done Feb 2016)	USA, Attended, OnlinePin, OfflinePin, Sig
Ingenico RBA family	5.26/1C	8.4+	Paymentech Tampa 3.2.1 (Done Apr 2018)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless, PIN ByPass
Elavon				
Ingenico RBA family	5.26/1C	8.4+	Elavon Encompass 3.1.0 (Jan 2018)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless, PIN ByPass
First Data				
Ingenico UPP (Tetra) family	8.38/1C	9.1+	First Data Cardnet or Nashville EDC 4.1.1	USA, Attended, OnlinePin, OfflinePin, Sig, EMV

Device	EMVKern/Conf	UniTerm	Module Version	Config
			(Done Dec 2018)	Debit, EMV Contactless, PIN ByPass
BBPos Chipper 2X BT	1.1/4C	9.1+	First Data Cardnet or Nashville EDC 4.1.1 (Done Dec 2018)	USA, Attended, NoCVM Only, EMV Contactless
IDTech Augusta	1.10.037/5C	9.1+	First Data Cardnet or Nashville EDC 4.1.1 (Done Dec 2018)	USA, Attended, Sig
IDTech Augusta + Spectrum Pro	1.10.037/4C	9.1+	First Data Cardnet or Nashville EDC 4.1.1 (Done Dec 2018)	USA, Unattended, NoCVM Only
IDTech Augusta + Spectrum Pro w/PinPad	1.10.037/4C	9.1+	First Data Cardnet or Nashville EDC 4.1.1 (Done Dec 2018)	USA, Unattended, OnlinePin, OfflinePin, EMV Debit
Ingenico RBA (Telium2) family	5.26/1C	8.4+	First Data Cardnet or Nashville EDC 4.1.1 (Done Dec 2017)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless, PIN ByPass
Ingenico RBA (Telium2) family	5.26/10C	8.4+	First Data Cardnet or Nashville EDC 4.1.1 (Done Dec 2017)	USA, Attended, Sig, EMV Debit, EMV Contactless, PIN ByPass
Ingenico RBA (Telium2) family	5.26/11C	8.4+	First Data Cardnet or Nashville EDC 4.1.1	USA, Unattended, OnlinePin, OfflinePin, EMV Debit, EMV

Device	EMVKern/Conf	UniTerm	Module Version	Config
			(Done Jan 2018)	Contactless, PIN ByPass
Ingenico RBA (Telium2) family	5.26/3C	8.4+	First Data Cardnet or Nashville EDC 4.1.1 (Done Jan 2018)	USA, Unattended, NoCVM only, EMV Debit, EMV Contactless
Ingenico RBA (Telium2) family	4.67/1C	8.3+	First Data Cardnet or Nashville EDC 4.1.1 (Done Feb 2017)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless
Ingenico RBA (Telium2) family	4.67/1C	8.0+	First Data Cardnet or Nashville EDC 4.1.0 (Done Jan 2016)	USA, Attended, OnlinePin, OfflinePin, Sig
Verifone vx805	6.2.0/1C	8.0+	First Data Cardnet or Nashville EDC 4.1.0 (Done Feb 2016)	USA, Attended, OnlinePin, OfflinePin, Sig
Global Payments				
Ingenico RBA family	4.67/1C	8.0+	Global Payments East 3.0.0 (Done Nov 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Verifone vx805	6.2.0/1C	8.0+	Global Payments East 3.0.0 (Done Nov 2015)	USA, Attended, OnlinePin, OfflinePin, Sig
Heartland Payment Systems				
Ingenico RBA family	5.26/1C	9.0+	Heartland Payment Systems 2.1.0 (Done Jan 2019)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV

Device	EMVKern/Conf	UniTerm	Module Version	Config
				Contactless, PIN ByPass
Moneris				
Ingenico iUP250 TCPX	5.38/5C	9.4+	Moneris SPDH 2.1.0 (Done June 2019)	Canada, Unattended, OfflinePin, EMV Debit, EMV Contactless
Tsys				
Ingenico RBA family	5.26/1C	9.4+	TSYS (aka Vital/VisaNet) 4.1.0 (Done June 2019)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless, PIN ByPass
Ingenico RBA family	4.67/1C	8.0+	TSYS (aka Vital/VisaNet) 3.0.0 (Done Feb 2016)	USA, Attended, OnlinePin, OfflinePin, Sig
Vantiv				
Ingenico RBA family	5.26/1C	8.4+	Vantiv/ FifthThird 610 3.0.0 (Done August 2017)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless, PIN ByPass
Ingenico RBA family	4.67/1C	8.0+	Vantiv/ FifthThird 610 2.1.0 (Done Jan 2016)	USA, Attended, OnlinePin, OfflinePin, Sig
WorldPay				
Ingenico RBA family	5.26/1C	8.4+	RBS WorldPay TCMP 2.1.0 (Done August 2017)	USA, Attended, OnlinePin, OfflinePin, Sig, EMV Debit, EMV Contactless, PIN ByPass
Ingenico RBA family	4.67/1C	8.0+	RBS WorldPay TCMP 2.0.0	USA, Attended, OnlinePin,

Device	EMVKern/Conf	UniTerm	Module Version	Config
			(Done Feb 2016)	OfflinePin, Sig

12.2 Configuration Definitions

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

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

Key	Description
USA	Certified for use in the United States
Canada	Certified for use in Canada
Attended	The environment is monitored by a clerk such as Retail, Restaurant, or Lodging. Not usable in a Kiosk environment such as a parking meter or gas pump.
Unattended	The environment is NOT monitored by a clerk, for use in kiosk type environments.
OnlinePin	An encrypted PIN can be obtained from a cardholder and sent to the processor with the transaction. When supporting Online PIN it is required that the device be injected with a `3DES DUKPT` PIN key specific to the processing institution in use prior to deployment by a merchant. 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 HMAC Algorithm

Keyed-Hashing for Message Authentication as described by RFC-2104

HMAC provides a way to check the integrity of information transmitted over or stored in an unreliable medium, and is a prime necessity in the world of open computing and communications. Mechanisms that provide such an integrity check based on a secret key are usually called "message authentication codes" (MAC). Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties.

Equation B.1. HMAC Mathmatical Definition

HMAC $(K,m) = H ((K \oplus opad) \# H ((K \oplus ipad) \# m))$

Where:

- `H` is a cryptographic hash function (SHA256 in UniTerm)

- `K` is a secret key padded to the right with extra zeros to the input block size of the hash function, or the hash of the original key if it's longer than that block size

- `m` is the message to be authenticated

- `#` denotes concatenation

- $\oplus$ denotes Exclusive OR (XOR)

- `opad` is the outer padding (0x5c5c5c...5c5c, one-block-long hexadecimal constant)

- `ipad` is the inner padding (0x363636...3636, one-block-long hexadecimal constant)

For more information please see RFC 2104: http://www.ietf.org/rfc/rfc2104.txt

Also, Wikipedia has a nice overview here: http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

---

C Pre-formatted Receipt Processing

Pre-formatted receipt processing has been added to simplify generation of compliant receipts with the rules dictated by the card brands. Data is output in a series of sections so that merchants may insert their own custom data in-between sections of brand-required data as they see fit.

C.1 Request parameters

The `u_rcpt` key/value pair is sent in the request to UniTerm to indicate whether or not to output a series of pre-formatted receipt blocks. This can also specifying format requirements. If set to `yes` it will simply use the receipt formatting configuration from the `uniterm.ini`.

Receipt data can be returned on these `u_action`'s:

- `txnrequest`
- `txnfinish`
- `tab` - when `u_tab=close`
- `standin` - when `u_standin=getresp`
- `passthrough`
- `passthroughmac`

When passing formatting configuration to `u_rcpt`, it will take the key/value pairs from the table below and encode them in a set of semi-colon separated key/value pairs such as:

```
u_rcpt=type=plain;line_len=24;use_merch_lang=no;line_break="\n"
```

or

```
type=plain|html
```

Receipt configuration options:

key	value
type	Possible values: • `plain` - Plain Text (default) • `html` - HTML. Needs style sheet applied. • `xml` - XML. Suitable for XSLT transformations. Typically used to generate complex HTML when CSS alone is not capable of providing the desired formatting. • `json` - JSON. Typically used for easy manipulation with Javascript. Values can be specified pipe-delimited (`|`) if more than one receipt output format is desired. When specifying more than one type, the response keys will indicate the format in the key name, but if only one type is specified, the type is omitted.
line_len	Only relevant for `type=plain`. Number of characters per line. Default is 24.

`line_break`	Only relevant for `type=plain`. Character sequence for use for newlines. Default is \r\n
`use_merch_lang`	True/False. Use the merchant's selected language rather than the cardholder's language for the receipt. Default is True.

C.2 Supported Languages

Currently Supported Languages:

- `en`: English
- `fr`: French
- `es`: Spanish
- `de`: German
- `it`: Italian

C.3 Response Data

The below data elements are ordered in the recommended order for each receipt type (with the exception for Moneris (`SPDH`), please see note below). Failure to use the provided order may result in a non-compliant receipt format. Some response fields may not be sent back on some transaction or response types.

Note: When using multiple types the response will have `u_rcpt` blocks denoting the type after `u_rcpt`. E.g. `u_rcpt_html_`, `u_rcpt_plain_`. If only one type is specified the type will not be included in the `u_rcpt` key name, the position of the receipt type is denoted by `*` in the key names if used.

Customer receipt blocks:

key	value
`u_rcpt_*cust_merch_info`	Merchant information on file such as name, address, and phone number.
`u_rcpt_*cust_type`	Transaction Type information. Most of the time it is emitted with a different section, except for Moneris receipt formatting.
`u_rcpt_*cust_reference`	Information about the transaction such as order number, authorization number, time/date and other identifying information.
`u_rcpt_*cust_money`	Transaction amount, balance, and other monetary information.
`u_rcpt_*cust_disposition`	Outcome of the transaction. Includes authorization mode, PIN validation, and other information about the transaction status.
`u_rcpt_*cust_emv`	Brand-required EMV data and tags.
`u_rcpt_*cust_notice`	Notice about receipt retention and indicate it is a customer copy.

> Note: Moneris (`SPDH`) customer receipts require blocks to be output in the below order:
>
> - `u_rcpt_*cust_merch_info`
> - `u_rcpt_*cust_type`
> - `u_rcpt_*cust_money`
> - `u_rcpt_*cust_reference`
> - `u_rcpt_*cust_disposition`
> - `u_rcpt_*cust_notice`

Merchant receipt blocks:

key	value
`u_rcpt_*merch_merch_info`	Merchant information on file such as name, address, and phone number.
`u_rcpt_*merch_type`	Transaction Type information. Most of the time it is emitted with a different section, except for Moneris receipt formatting.
`u_rcpt_*merch_reference`	Information about the transaction such as order number, authorization number, time/date and other identifying information.
`u_rcpt_*merch_money`	Transaction amount, balance, and other monetary information.
`u_rcpt_*merch_disposition`	Outcome of the transaction. Includes authorization mode, PIN validation, and other information about the transaction status.
`u_rcpt_*merch_signature`	Will be present when a signature is necessary, or has been electronically captured.
`u_rcpt_*merch_emv`	Brand-required EMV data and tags.
`u_rcpt_*merch_notice`	Notice about receipt retention and indicate it is a merchant copy.

> Note: Moneris (`SPDH`) merchant receipts require blocks to be output in the below order:
>
> - `u_rcpt_*merch_merch_info`
> - `u_rcpt_*merch_type`
> - `u_rcpt_*merch_money`
> - `u_rcpt_*merch_reference`
> - `u_rcpt_*merch_disposition`
> - `u_rcpt_*merch_notice`

C.3.1 Plain Text format

The sections returned as per the prior section are intended to simply be concatenated together in the order documented to generate a fully compliant receipt. No additional processing is necessary.

C.3.2 HTML format

UniTerm is capable of returning transaction receipt data as HTML. It will return the set of receipt-specific key-value pairs as documented in the prior section with the transaction response, where the values are HTML blocks containing a section of receipt content, and the keys indicate which section is contained in the value.

Each HTML block is wrapped in a `div` container element with a class of `rcpt_block` and an `id` indicating the block's purpose. Standalone data values in the block are wrapped in `div` elements with the class `rcpt_value` and an `id` indicating what the data is (e.g. "merch_name", "merch_phone", "action", etc). Labeled data values in the block are grouped together in `div` elements with the class `rcpt_group`; the label is contained in a `div` with the class `rcpt_group_label` and the value is contained in a `div` with the class `rcpt_group_val`.

All classes and `ids` used are documented in the following sections.

C.3.2.1 Classes

Class	Purpose
`rcpt_block`	Container for a block of receipt data
`rcpt_val`	An unlabeled item of receipt data. Direct descendant of `rcpt_block`
`rcpt_group`	Container for labeled receipt data. Direct descendant of `rcpt_block`
`rcpt_group_label`	A label for an item of receipt data. Direct descendant of `rcpt_group`
`rcpt_group_val`	A labeled item of receipt data. Direct descendant of `rcpt_group`

C.3.2.2 IDs

ID	Purpose
`rcpt_merch_info`	Block containing merchant information
`merch_name`	Merchant's name
`merch_addr1`	Merchant's street address
`merch_addr2`	Merchant's unit/suite number
`merch_addr3`	Merchant's city, state, and zip code
`merch_phone`	Merchant's phone number
`merch_email`	Merchant's email address
`merch_url`	Merchant's website URL
`rcpt_reference`	Block containing transaction reference information
`action`	The 'action' value for the transaction (sale, return, etc.)
`merch_id_group`	Container for `merch_id_label` and `merch_id`
`merch_id_label`	Label for merchant ID
`merch_id`	`Merchant ID`

`laneid_group`	Container for `laneid_label` and `laneid`
`laneid_label`	Label for lane ID
`laneid`	Lane ID
`cardtype`	The type of card used in the transaction (VISA, MC, etc.)
`entry_mode`	The transaction's entry mode
`acct_type_group`	Container for `acct_type_label` and `acct_type`
`acct_type_label`	Label for account type
`acct_type`	Account type
`account_group`	Container for `account_label` and `account`
`account_label`	Label for account number
`account`	Account number used in the transaction
`time_group`	Container for `time_label` and `time`
`time_label`	Label for transaction date/time
`time`	Date/time at which the transaction took place
`ordernum_group`	Container for `ordernum_label` and `ordernum`
`ordernum_label`	Label for order number
`ordernum`	Order number
`custref_group`	Container for `custref_label` and `custref`
`custref_label`	Label for customer reference number
`custref`	Customer reference number
`ttid_group`	Container for `ttid_label` and `ttid`
`ttid_label`	Label for TTID
`ttid`	TTID
`resp_code_group`	Container for `resp_code_label` and `resp_code`
`resp_code_label`	Label for response code
`resp_code`	Response code
`auth_group`	Container for `auth_label` and `auth`
`auth_label`	Label for authorization number
`auth`	Authorization number
`batch_group`	Container for `batch_label` and `batch`
`batch_label`	Label for batch number
`batch`	Batch number
`stan_group`	Container for `stan_label` and `stan`
`stan_label`	Label for STAN
`stan`	STAN (system trace audit number)
`rcpt_money`	Block containing monetary amounts from transaction

`amount_group`	Container for `amount_label` and `amount`
`amount_label`	Label for transaction amount
`amount`	Transaction amount
`tip_group`	Container for `tip_label` and `tip`
`tip_label`	Label for tip amount
`tip`	Tip amount
`tax_group`	Container for `tax_label` and `tax`
`tax_label`	Label for tax amount
`tax`	Tax amount
`cashbackamount_group`	Container for `cashbackamount_label` and `cashbackamount`
`cashbackamount_label`	Label for cash back amount
`cashbackamount`	Cash back amount
`authamount_group`	Container for `authamount_label` and `authamount`
`authamount_label`	Label for auth amount
`authamount`	Auth amount
`totalamount_group`	Container for `totalamount_label` and `totalamount`
`totalamount_label`	Label for total amount for transaction
`totalamount`	Total amount for transaction
`balance_group`	Container for `balance_label` and `balance`
`balance_label`	Label for customer balance amount
`balance`	Customer balance amount
`card_status`	Card status
`pin_status`	PIN status
`auth_result`	Transaction result (e.g. "Approved", "Declined")
`auth_mode_group`	Container for `auth_mode_label` and `auth_mode`
`auth_mode_label`	Label for authorization mode
`auth_mode`	Authorization mode
`printdata`	Print data
`rcpt_signature`	Block containing signature information
`line`	Signature line
`agreement`	Signature agreement
`sigcaptured`	Transaction signature capture status
`rcpt_emv`	Block containing EMV data
`name_group`	Container for `name_label` and `name`
`name_label`	Label for EMV name
`name`	EMV name

`aid_group`	Container for `aid_label` and `aid`
`aid_label`	Label for EMV AID
`aid`	EMV AID
`tvr_group`	Container for `tvr_label` and `tvr`
`tvr_label`	Label for EMV `TVR`
`tvr`	EMV `TVR`
`tsi_group`	Container for `tsi_label` and `tsi`
`tsi_label`	Label for EMV `TSI`
`tsi`	EMV `TSI`
`actype_group`	Container for `actype_label` and `actype`
`actype_label`	Label for EMV `AC type`
`actype`	EMV `AC type`
`ac_group`	Container for `ac_label` and `ac`
`ac_label`	Label for EMV Application Cryptogram
`ac`	EMV Application Cryptogram
`rcpt_notice`	Block containing notices for the customer
`message`	Notice message
`copy`	Whether receipt is a merchant copy or customer copy

C.3.2.3 Simple CSS styling example

```
 1  .rcpt_block {
 2    font-family: monospace;
 3    width: 15rem;
 4    display: grid;
 5    grid-template-columns: 1fr 1fr;
 6    grid-template-rows: auto;
 7    margin-bottom: 1rem;
 8  }
 9  .rcpt_val, .rcpt_group {
10    grid-column: 1/3;
11    margin-bottom: .5rem;
12  }
13  .rcpt_val {
14    text-align: center;
15    display: block;
16  }
17  .rcpt_group {
18    grid-column: 1/3;
19    display: flex;
20    justify-content: space-between;
21  }
22  .rcpt_group_label:after {
23    content: ":";
24  }
25  #action, #auth_result, .rcpt_group_label {
```

```
26    text-transform: uppercase;
27  }
28  #cardtype, #entry_mode, #auth_group, #batch_group {
29    grid-column: auto / span 1;
30  }
31  #cardtype {
32    text-align: left;
33  }
34  #entry_mode, #batch_group {
35    text-align: right;
36  }
37  #auth_group, #batch_group {
38    display: inline-block;
39  }
```

C.3.2.4 Complex CSS styling example

```
 1  * {
 2    margin: 0;
 3    padding: 0;
 4  }
 5  main {
 6    padding: 1rem;
 7    display: grid;
 8    grid-template-columns: 1fr 1fr;
 9    grid-template-rows: auto;
10    position: relative;
11  }
12  #rcpt_merch_info {
13    font-size: 1.2rem;
14  }
15  #rcpt_reference {
16    font-size: .9rem;
17    display: grid;
18    grid-template-columns: 1fr 1fr 1fr;
19  }
20  #rcpt_merch_info, #rcpt_reference, #rcpt_money {
21    padding: 1rem;
22    margin: 1rem;
23    background-color: #F8F8F8;
24    border: 1px solid #E9E9E9;
25  }
26  #rcpt_merch_info, #rcpt_reference {
27    grid-column: auto / span 1;
28  }
29  #rcpt_merch_info:before, #rcpt_reference:before {
30    font-weight: bold;
31    display: block;
32    font-size: 1rem;
33    margin-bottom: .75rem;
34  }
35  #rcpt_merch_info:before {
36    content: "Store Information:";
37  }
38  #rcpt_reference:before {
39    content: "Transaction Details:";
40    grid-column: auto / span 3;
```

```
41    order: -4;
42  }
43  .rcpt_group_label:after {
44    content: ": ";
45  }
46  #rcpt_reference .rcpt_group {
47    display: flex;
48    justify-content: space-between;
49    padding: .25rem;
50    grid-column: auto / span 3;
51  }
52  #rcpt_reference .rcpt_val {
53    margin-bottom: .5rem;
54  }
55  #rcpt_reference .rcpt_group:nth-child(even) {
56    background-color: #E9E9E9;
57  }
58  #action, #auth_result {
59    text-transform: uppercase;
60  }
61  #action {
62    order: -3;
63  }
64  #cardtype {
65    order: -2;
66    text-align: center;
67  }
68  #entry_mode {
69    order: -1;
70    text-align: right;
71  }
72  #entry_mode:before {
73    content: "Entry Mode: ";
74  }
75
76  #rcpt_money {
77    grid-column: auto / span 2;
78    text-align: right;
79  }
80  #totalamount_group {
81    font-size: 1.4rem;
82    font-weight: 700;
83  }
84  #rcpt_disposition {
85    grid-column: auto / span 2;
86    text-align: center;
87    margin: 1rem;
88    padding: 1rem;
89    margin-top: 0;
90    padding-top: 0;
91  }
92  #auth_result {
93    font-weight: 700;
94    font-size: 1.2rem;
95  }
96  #rcpt_signature, #rcpt_notice {
97    grid-column: auto / span 2;
98    text-align: center;
```

```
 99    color: #777;
100    }
101    @media (max-width: 800px) {
102     #rcpt_merch_info, #rcpt_reference {
103      grid-column: auto / span 2;
104     }
105     #rcpt_merch_info:before, #rcpt_reference:before {
106      text-align: center;
107     }
108     #rcpt_merch_info {
109      text-align: center;
110     }
111     #rcpt_money {
112      text-align: center;
113     }
114     header {
115      padding: .5rem;
116      flex-direction: column;
117     }
118    }
```

C.3.3 XML and JSON format

For XML each block has a top level element with the block name, E.g. `merch_info`, `reference`.

JSON will not have the top level element name surrounding the data.

Block and elements:

- merch_info:
  - merch_name
  - merch_addr1
  - merch_addr2
  - merch_addr3
  - merch_phone
  - merch_email
  - merch_url

- reference:
  - action
  - merch_id
  - laneid
  - cardtype
  - rcpt_entry_mode
  - rcpt_acct_type
  - account
  - time
  - ordernum
  - custref
  - ttid
  - rcpt_issuer_resp_code

- `rcpt_resp_code`
- `auth`
- `batch`
- `stan`
- `rcpt_custom`
  - Custom/Undocumented items here

- `money:`
  - `amount`
  - `tip`
  - `tax`
  - `cashbackamount`
  - `totalamount`
  - `authamount`
  - `balance`

- `disposition:`
  - `card_status`
  - `pin_status`
  - `auth_result`
  - `auth_mode`
  - `printdata`

- `signature:`
  - `line`
  - `agreement`
  - `cardholdername`
  - `sigcaptured`

- `emv:`
  - `name`
  - `aid`
  - `tvr`
  - `tsi`
  - `actype`
  - `ac`

- `notice:`
  - `message`
  - `copy`

D EMV Receipt Requirements (for Manual Receipt formatting)

The UniTerm application never directly generate receipts, but it can emit pre-formatted blocks for receipts. This section relates to complete manual receipt generation not using pre-formatted blocks. 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).

As of UniTerm v9.1, additional receipt generation helpers were added and are recommended for use rather than manually formatting a receipt as per this section. Please see Appendix C.

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.

D.1 Receipt content

D.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` - If using the `returnbin` feature, the first 6 digits may also be returned in the clear. It is the integrator's responsibility to ensure only the last 4 digits of the account number are provided in the clear.
- Transaction reference info
  - Date and time - `rcpt_host_ts` or `timestamp`
  - Identifier - request parameters `ordernum` or `ptrannum`
  - Additional identifiers (optional) - request parameters such as `custref`
  - `ttid` (optional) - either request or response `ttid`
  - Batch number - `batch`
  - Auth number (if authorized) - `auth`
  - Trace information - `stan`
  - Processor response code (some processors may require this) - `rcpt_resp_code`
  - Issuer response code (some processors may require this) - `rcpt_issuer_resp_code`
- Processor specific custom data - see `rcpt_custom`
- Monetary amounts
  - Tip - Request parameter `examount`
  - Tax - Request parameter `tax`
  - Cash back - Request parameter `cashbackamount`
  - Authorized Amount - `authamount` if returned, otherwise request parameter `amount`
  - Balance - `balance`
- Transaction disposition
  - Card disposition - See Card Disposition documentation
  - Partial Approval Indicator - if `authamount` returned
  - Overall disposition (approved/declined) - `code`
- Additional Print Data - `printdata`, Additional data meant to be printed on the receipt as provided by the processor. Often used for gift/loyalty programs.
- Cardholder Verification
  - Signature line (if necessary) - `u_need_signature=yes`
  - Cardholder Name, centered under signature line if a signature line is shown (optional) - `cardholdername`
  - PIN Entry - Verified by PIN or PIN Bypassed if appropriate
- 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

D.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` • `skipped` - Used for some contactless transactions below the `CVM` floor. • `cdcvm` - Card Holder Device verification. The user's device (usually smart phone) validated the user's identity. • `unknown` For `"pin"` and `"pinsig"` the receipt should say "VERIFIED BY PIN". For `"sig"` a signature should be captured.
rcpt_emv_pinbypass	Returned as true if the cardholder explicitly requested the pin to be bypassed, false or not provided at all otherwise. Receipt should say "PIN BYPASSED".
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.

D.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
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 `authhamount` is returned and is not equal to requested amount, should say: "TRANSACTION PARTIALLY APPROVED"
- When `code` is `CALL`, should say: "CALL ISSUER"

D.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.

D.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

D.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.

Note: UniTerm supports multiple languages for receipts and display, however `Moneris` prohibits use of any languages other than English and French.

D.7 Receipt Examples

Monetra Technologies 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.

D.7.1 EMV Insert, Signature Required

D.7.1.1 UniTerm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXXX0119
auth	152013
batch	1
cardholdername	VISA ACQUIRER TEST/CARD 01
cardlevel	VISA_TRADITIONAL
cardtype	VISA
code	AUTH
item	27
language	en
merch_addr1	123 STREET NAME
merch_addr2	CITY, STATE ZIP
merch_id	1834
merch_name	MERCHANT NAME
merch_phone	(888) 555-1234
merch_proc	GLOBALPAY
msoft_code	INT_SUCCESS
pclevel	0
phard_code	SUCCESS
rcpt_custom	REC #:000027,TRN REF #:355724280069888,VAL CODE:BBCD
rcpt_emv_ac	8F73ED36C8F2C099
rcpt_emv_actype	TC
rcpt_emv_aid	A0000000031010
rcpt_emv_cvm	sig
rcpt_emv_name	CREDITO DE VISA
rcpt_emv_tsi	F800
rcpt_emv_tvr	0280008000
rcpt_entry_mode	C

rcpt_host_ts	092215174640
rcpt_issuer_resp_code	00
rcpt_resp_code	000
stan	378222
timestamp	1442944083
ttid	29
u_errorcode	SUCCESS
u_need_signature	yes
verbiage	AP

D.7.1.2 Example Receipt

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

           SALE

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

AMOUNT:             1.00

        APPROVED

SIGNATURE

X_____
 VISA ACQUIRER TEST/CARD
          01

CARDHOLDER WILL PAY CARD
   ISSUER ABOVE AMOUNT
 PURSUANT TO CARDHOLDER
       AGREEMENT

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

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS
```

```
      MERCHANT COPY

------

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

            SALE

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

AMOUNT:             1.00

         APPROVED

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

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

D.7.2 EMV Insert, PIN Verified

D.7.2.1 UniTerm Response Data

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

---

timestamp	1442942662
ttid	7
u_errorcode	SUCCESS
verbiage	AP

D.7.2.2 Example Receipt

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


           SALE

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

AMOUNT:          337.00


     VERIFIED BY PIN


         APPROVED

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

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY


------

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


           SALE

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

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

AMOUNT:            337.00

     VERIFIED BY PIN

        APPROVED

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

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

D.7.3 EMV Insert, No CVM

D.7.3.1 UniTerm Response Data

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

ttid	79
u_errorcode	SUCCESS
verbiage	AP

D.7.3.2 Example Receipt

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

            SALE

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

AMOUNT:          62.00

        APPROVED

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008000
TSI F800
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY

------

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

            SALE

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

```
AMOUNT:              62.00

        APPROVED

AMERICAN EXPRESS
AID A000000025010801
TVR 0000008000
TSI F800
TC 5C221DC28EB72FCF

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

D.7.4 EMV Insert, Card Decline

D.7.4.1 UniTerm Response Data

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

D.7.4.2 Example Receipt

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


        SALE

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

AMOUNT:            22.00
```

```
        DECLINED BY CARD
         VERIFIED BY PIN

             DECLINED

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

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

        MERCHANT COPY


------

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

               SALE

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

AMOUNT:           22.00

    DECLINED BY CARD
      VERIFIED BY PIN

          DECLINED

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

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

        CUSTOMER COPY
```

D.7.5 EMV Insert, Card Removed (Decline)

D.7.5.1 UniTerm Response Data

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

D.7.5.2 Example Receipt

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

         SALE

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

AMOUNT:           1.00

     CARD REMOVED

       DECLINED


 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

     MERCHANT COPY

------

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

         SALE

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

```
AMOUNT:               1.00

      CARD REMOVED

         DECLINED


 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

D.7.6.1 UniTerm Response Data

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

ttid	10
u_errorcode	SUCCESS
verbiage	APPROVED

D.7.6.2 Example Receipt

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


          SALE

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

AMOUNT:             5.01

     VERIFIED BY PIN


         APPROVED

Interac
AID A0000002771010
TVR 8000008000
TSI 7800
TC 882D8427A268E214


 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      MERCHANT COPY


------

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


          SALE

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

```
AMOUNT:                 5.01

     VERIFIED BY PIN

         APPROVED

Interac
AID A0000002771010
TVR 8000008000
TSI 7800
TC 882D8427A268E214

 IMPORTANT - RETAIN THIS
  COPY FOR YOUR RECORDS

      CUSTOMER COPY
```

D.7.7 EMV Contactless, Interac Flash Decline

D.7.7.1 UniTerm Response Data

PARAMETER	VALUE
account	XXXXXXXXXXXXXXXX1311
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

D.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: XXXXXXXXXXXXXXXX1311
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
```

D.7.8 EMV Contactless, Decline

D.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 *

D.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
```

E UniTerm Code Examples

E.1 Microsoft C# using libmonetra

```
 1  /* UniTerm example program in C#
 2   *
 3   * Depends on the libmonetra C# .Net native API
 4   *
 5   * Implemented based on the UniTerm Guide in conjunction with the
 6   * Monetra Client Interface Protocol Specification
 7   *
 8   * Please contact support@monetra.com with any questions
 9   */
10  using System;
11  using System.Collections;
12  using System.Diagnostics;
13  using System.IO;
14  using System.Text;
15  using System.Threading;
16  using libmonetra;
17
18  /* NOTE: if compiling with Mono, you can use
19   *        gmcs /unsafe utest.cs libmonetra.cs
20   */
21
22  class UTest {
23   /* Uniterm Connectivity Information
24    * NOTE: this is the default, it is possible to change, but 99%
25    *       of deployments will probably use this Uniterm information
26    *       as-is
27    */
28   private const string uniterm_host    = "localhost";
29   private const int    uniterm_port    = 8123;
30
31   /* Authentication information
32    * NOTE: This information corresponds with the public test server
33    *       at testbox.monetra.com:8665 */
34   private const string monetra_user    = "test_retail:public";
35   private const string monetra_pass    = "publ1ct3st";
36
37
38   static string uniterm_path()
39   {
40    switch (Environment.OSVersion.Platform) {
41     case PlatformID.Win32NT:
42     case PlatformID.Win32S:
43     case PlatformID.Win32Windows:
44     case PlatformID.WinCE:
45      return "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe";
46     default:
47      return "/usr/local/uniterm/bin/uniterm";
48    }
49   }
50
51   /*! Function to launch the Uniterm from the current process.
```

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

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

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

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

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

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

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

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

```
137      rdr.Close();
138    } catch (System.Net.WebException e)  {
139     response["code"]       = "DENY";
140     response["u_errorcode"] = "CONN_ERROR";
141     response["verbiage"]   = "Connection to " + url + " failed: " +
142                              e.Message;
143    return response;
144   }
145   XmlDocument    xmldoc    = new XmlDocument();
146   xmldoc.LoadXml(xmlout);
147
148   XmlNodeList    trans    = xmldoc.DocumentElement.
149                            SelectSingleNode("Resp").ChildNodes;
150   foreach (XmlNode kv in trans) {
151    response[kv.Name] = kv.InnerText;
152   }
153   return response;
154  }
155
156
157  /*! Request a ttransaction from Uniterm as documented in the Monetra
158   *  Uniterm Guide.  The Key/Value pair params are a combination of the
159   *  Parameters as documented in the Uniterm Guide and the Monetra Client
160   *  Interface Protocol Spec.
161   * \param[in] mparams   Array of key/value parameters to send to Uniterm
162   * \return Hashtable of string key/value pairs from response.  Please refer
163   *         to the Uniterm Guide and Monetra Client Interface Protocol
164   *         specification for the applicable list based on the action being
165   *         performed.  "code" and "u_errorcode" are always guaranteed to
166   *         be returned.
167   */
168  static Hashtable uniterm_sendrequest(Hashtable mparams)
169  {
170   string XML;
171
172   XML = "<MonetraTrans>" +
173         "<Trans identifier='1'>";
174
175   /* For each item in the params hashtable, add it to the transaction */
176   foreach (DictionaryEntry kv in mparams) {
177    XML = XML + "<" + (String)kv.Key + ">" + (String)kv.Value + "</" +
178         (string)kv.Key + ">";
179   }
180
181   XML = XML + "</Trans></MonetraTrans>";
182
183   return uniterm_https_post(uniterm_host, uniterm_port, XML);
184  }
185
186
187  /*! Tell Uniterm to shutdown.  Since we start it up,
188   *  we should make sure we turn it off prior to exiting otherwise
189   *  the user will be prompted with an error message stating
190   *  Uniterm is already running on the next execution
191   *  of this application!
192   */
193  static void uniterm_shutdown()
194  {
```

```
195     Hashtable mparams = new Hashtable();
196     mparams["u_action"] = "shutdown";
197     uniterm_sendrequest(mparams);
198   }
199
200
201   /*! Main entry point to this application to be executed */
202   static void Main()
203   {
204    Hashtable response;
205
206    /* Step1: Launch Uniterm */
207    uniterm_launch();
208    MessageBox.Show("Uniterm Launched");
209
210
211    /* Step2: Send txnrequest to Uniterm */
212    Hashtable mparams = new Hashtable();
213    /* Append the parameters for the ticket request as per the Monetra
214     * Uniterm Guide, section 4 */
215    mparams["username"]     = monetra_user;
216    mparams["password"]     = monetra_pass;
217    mparams["u_action"]      = "txnrequest";
218    mparams["u_devicetype"] = "ingenico_rba";
219    mparams["u_device"]     = "HID";
220
221    /* Append the parameters for the transaction that will also get passed
222     * to Monetra such as the 'action', 'amount', etc. as described in the
223     * Monetra Client Interface Protocol Specification */
224    mparams["action"]  = "sale";
225    mparams["amount"]   = "12.00";
226    mparams["ordernum"] = "123456";
227    mparams["comments"] = "u_txnrequest";
228
229    response = uniterm_sendrequest(mparams);
230    string resultMsg = "";
231    if (String.Compare((string)response["code"], "AUTH", true) != 0) {
232     resultMsg = "Transaction failed.\r\n";
233    } else {
234     resultMsg = "Transaction SUCCESSFUL!\r\n";
235    }
236
237    /* Print out all the response key/value pairs ... */
238    foreach (DictionaryEntry kv in response) {
239     resultMsg = resultMsg + (string)kv.Key + " = " + (string)kv.Value +
240                "\r\n";
241    }
242
243    MessageBox.Show(resultMsg);
244
245    /* NOTE: No real reason to exit here ... we could just keep running
246     *       Step 2 all day long as long.
247     *       No reason to keep starting/stopping Uniterm.
248     */
249
250    /* Step3: Cleanup */
251    uniterm_shutdown();
252
```

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

E.3 Java using libmonetra

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

```
49    static void uniterm_launch()
50    {
51     try {
52      Process p = new ProcessBuilder(uniterm_path()).start();
53     } catch (java.io.IOException e) {
54      System.out.println(e.getMessage());
55      System.exit(1);
56     }
57     /* Make sure Uniterm is ready before returning,
58      * Sleep 1000ms (1s) */
59     try {
60      Thread.sleep(1000);
61     } catch (InterruptedException e) {
62     }
63    }
64
65
66    /*! Function to connect to an endpoint which uses the standard 'monetra'
67     * style protocol (so either Monetra itself, or Uniterm)
68     * \param[in]  host     Resolvable hostname or IP address to connect to
69     * \param[in]  port     Port associated with hostname to establish an SSL
70     *                      connection to
71     * \param[out] errorstr Textual error message if returns null
72     * \return Initialized connection class on success. null on failure
73     */
74    static MONETRA uniterm_connect_host(String host, int port,
75                                        StringBuilder errorstr)
76    {
77     /* Initialize the Class */
78     MONETRA conn = new MONETRA("");
79
80     errorstr.setLength(0);
81
82     /* We always want to use an SSL connection to Monetra and Uniterm */
83     conn.SetSSL(host, port);
84
85     /* Do not verify the SSL certificate, Monetra and Uniterm
86      * use self-signed certificates by default which cannot be validated.
87      * The connection is still encrypted, the endpoint just isn't strictly
88      * validated */
89     conn.VerifySSLCert(0);
90
91     /* This makes it so TransSend() will block until a response is
92      * received from Monetra.  Simplifies the API since we will never
93      * have more than one outstanding transaction per connection in
94      * this application */
95     conn.SetBlocking(1);
96
97     /* Connect! */
98     if (conn.Connect() == 0) {
99      errorstr.append(conn.ConnectionError());
100     return null;
101    }
102
103    return conn;
104   }
105
106
```

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

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

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

E.4 PHP using libmonetra

```php
 1  <?php
 2  /* UniTerm example program in PHP
 3   *
 4   * Depends on the libmonetra PHP native API
 5   *
 6   * Implemented based on the UniTerm Guide in conjunction with the
 7   * Monetra Client Interface Protocol Specification
 8   *
 9   * Please contact support@monetra.com with any questions
10   */
11  error_reporting(E_ALL);
12  require_once("libmonetra.php");
```

```php
13
14
15
16  /* Uniterm Connectivity Information
17   * NOTE: this is the default, it is possible to change, but 99%
18   *       of deployments will probably use this uniterm information
19   *       as-is
20   */
21  $uniterm_host      = "localhost";
22  $uniterm_port      = 8123;
23
24  /* Authentication information
25   * NOTE: This information corresponds with the public test server
26   *       at testbox.monetra.com:8665 */
27  $monetra_user   = "test_retail:public";
28  $monetra_pass   = "publ1ct3st";
29
30
31  /* Sets the path of the Uniterm executable.  Currently using
32   * the default locations */
33  if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
34   /* Windows path */
35   $uniterm_path        = "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe";
36  } else {
37   /* Unix path */
38   $uniterm_path        = "/usr/local/uniterm/bin/uniterm";
39  }
40
41
42  /*! Function to launch Uniterm from the current process.
43   *  If we don't launch it from the current process, it won't be given
44   *  focus! (at least on Windows this is true, until the first
45   *  manual focus is performed by an end-user) */
46  function uniterm_launch()
47  {
48   global $uniterm_path;
49   if (class_exists("COM")) {
50    /* Must be running windows */
51    $WshShell = new COM("WScript.Shell");
52    $oExec    = $WshShell->Run('"' . $uniterm_path . '"', 10, false);
53   } else {
54    /* Must be on a Unix system */
55    system("'" . $uniterm_path . "'" . " > /dev/null 2>&1 &");
56   }
57
58   /* Make sure Uniterm is ready before returning,
59    * sleep 2s */
60   sleep(2);
61  }
62
63
64  /*! Function to connect to an endpoint which uses the standard 'monetra'
65   * style protocol (so either Monetra itself, or Uniterm)
66   * \param[in]  host     Resolvable hostname or IP address to connect to
67   * \param[in]  port     Port associated with hostname to establish an SSL
68   *                      connection to
69   * \param[out] errorstr Textual error message if returns null
70   * \return Initialized connection on success. null on failure
```

```php
 71   */
 72  function uniterm_connect_host($host, $port, &$errorstr)
 73  {
 74   /* Initialize the Connection */
 75   $conn = M_InitConn();
 76
 77   $errorstr = "";
 78
 79   /* We always want to use an SSL connection to Monetra and Uniterm */
 80   M_SetSSL($conn, $host, $port);
 81
 82   /* Do not verify the SSL certificate, Monetra and Uniterm
 83    * use self-signed certificates by default which cannot be validated.
 84    * The connection is still encrypted, the endpoint just isn't strictly
 85    * validated */
 86   M_VerifySSLCert($conn, false);
 87
 88   /* This makes it so TransSend() will block until a response is
 89    * received from Monetra.  Simplifies the API since we will never
 90    * have more than one outstanding transaction per connection in
 91    * this application */
 92   M_SetBlocking($conn, true);
 93
 94   /* Connect! */
 95   if (!M_Connect($conn)) {
 96    $errorstr = M_ConnectionError($conn);
 97    return null;
 98   }
 99
100   return $conn;
101  }
102
103
104  /*! Wrapper function to connect to Uniterm
105   * \param[out] errorstr Textual error message if returns null
106   * \return Initialized connection on success. null on failure
107   */
108  function uniterm_connect(&$errorstr)
109  {
110   global $uniterm_host, $uniterm_port;
111
112   $myerror = "";
113   $conn = uniterm_connect_host($uniterm_host, $uniterm_port, &$myerror);
114   if ($conn == null) {
115    $errorstr = "Connection to Uniterm Failed: " . $myerror;
116   }
117   return $conn;
118  }
119
120
121  /*! Request a transaction from Uniterm as documented in the Uniterm Guide.
122   *  The Key/Value pair params are a combination of the Parameters as
123   *  documented in the Uniterm Guide and the Monetra Client Interface Protocol
124   *  Spec.
125   * \param[in] uniterm_conn  Initialized connection to Uniterm as returned by
126   *                          uniterm_connect()
127   * \param[in] params        Array of key/value parameters to send to Uniterm
128   *
```

```php
129    * \return Array of string key/value pairs from response.  Please refer to the
130    *          Uniterm Guide and Monetra Client Interface Protocol specification
131    *          for the applicable list based on the action being performed.
132    *          "code" and "u_errorcode" are always guaranteed to be returned.
133    */
134   function uniterm_sendrequest($uniterm_conn, $params)
135   {
136    $response = array();
137
138    /* Request a new transaction from libmonetra */
139    $id = M_TransNew($uniterm_conn);
140
141    /* For each item in the params array, add it to the transaction */
142    foreach ($params as $key => $value) {
143     M_TransKeyVal($uniterm_conn, $id, $key, $value);
144    }
145
146    /* Send the request to the Uniterm.  It will not return until a
147     * response is available, or a disconnect is detected */
148    if (!M_TransSend($uniterm_conn, $id)) {
149     /* Disconnect detected, return an appropriate error condition!
150      * This should really never happen though... */
151     $response["code"]        = "DENY";
152     $response["u_errorcode"] = "CONN_ERROR";
153     $response["verbiage"]    = "Connection to Uniterm failed: " .
154                                 M_ConnectionError($uniterm_conn);
155     return $response;
156    }
157
158    /* Save the response parameters from the Uniterm into a HashTable
159     * as our function prototype states. */
160    $keys = M_ResponseKeys($uniterm_conn, $id);
161    foreach ($keys as $value) {
162     $response[$value]        = M_ResponseParam($uniterm_conn, $id, $value);
163    }
164
165    /* Free up some memory by purging unneeded data */
166    M_DeleteTrans($uniterm_conn, $id);
167
168    return $response;
169   }
170
171
172   /*! Tell Uniterm to shutdown.  Since we start it up,
173    *  we should make sure we turn it off prior to exiting otherwise
174    *  the user will be prompted with an error message stating the
175    *  Uniterm is already running on the next execution
176    *  of this application!
177    * \param[in] uniterm_conn Initialized connection to Uniterm
178    *                         as returned by uniterm_connect()
179    */
180   function uniterm_shutdown($uniterm_conn)
181   {
182    uniterm_sendrequest($uniterm_conn, array("u_action" => "shutdown"));
183   }
184
185
186
```

```
187   /* CODE TO EXECUTE ... */
188
189   $errorstr = "";
190
191   /* Step1: Launch Uniterm */
192   uniterm_launch();
193   echo "Uniterm Launched\r\n";
194
195   /* Step2: Connect to Uniterm */
196   $uniterm_conn = uniterm_connect(&$errorstr);
197   if ($uniterm_conn == null) {
198    echo "Failure: " . $errorstr . "\r\n";
199    return;
200   }
201
202   echo "Connected to Uniterm\r\n";
203
204
205   /* Step3: Send a txnrequest to the Uniterm */
206   $params  = array();
207
208   /* Append the parameters for the txnrequest */
209   $params["username"]     = $monetra_user;
210   $params["password"]     = $monetra_pass;
211   $params["u_action"]     = "txnrequest";
212   $params["u_devicetype"] = "ingenico_rba";
213   $params["u_device"]     = "HID";
214
215   /* Append the parameters for the transaction that will also get passed to
216    * Monetra such as the 'action', 'amount', etc. as described in the Monetra
217    * Client Interface Protocol Specification */
218   $params['action']  = 'sale';
219   $params['amount']   = '12.00';
220   $params['ordernum'] = '123456';
221   $params['comments'] = 'u_txnrequest';
222
223   $response = uniterm_sendrequest($uniterm_conn, $params);
224   if (strcasecmp($response["code"], "AUTH") != 0) {
225    echo "Transaction Failed.\r\n";
226   } else {
227    echo "Transaction SUCCESSFUL!\r\n";
228   }
229
230   /* Print out all the response key/value pairs ... */
231   foreach ($response as $key => $value) {
232    echo "\t" . $key . " = " . $value . "\r\n";
233   }
234
235   /* NOTE: No real reason to exit here ... we could just keep running
236    *       Step 3 all day long as long as you keep the uniterm_conn handle.
237    *       No reason to keep disconnecting and reconnecting, or
238    *       starting/stopping Uniterm.
239    */
240
241   /* Step4: Cleanup */
242   uniterm_shutdown($uniterm_conn);
243
244   /* Connections will be automatically closed when the uniterm_conn
```

```
245   * initialized connection is closed by the destructor/garbage collector */
246
247  ?>
248
249
```

E.5 Microsoft VB.Net using libmonetra

```
 1  ' UniTerm example program in VB.Net
 2  '
 3  ' Depends on the libmonetra C# .Net native API (DLL)
 4  '
 5  ' Implemented based on the UniTerm Guide in conjunction with the
 6  ' Monetra Client Interface Protocol Specification
 7  '
 8  ' Please contact support@monetra.com with any questions
 9
10  Option Explicit On
11  Option Strict On
12
13  Imports System
14  Imports System.Collections
15  Imports System.Diagnostics
16  Imports System.Threading
17  Imports libmonetra
18
19  ' On unix, compile using:
20  '   gmcs /target:library /unsafe libmonetra.cs
21  '   vbnc2 -r:libmonetra.dll utest.vb
22
23  Module Module1
24   ' Uniterm Connectivity Information
25   ' NOTE: this is the default, it is possible to change, but 99%
26   '       of deployments will probably use this Uniterm information
27   '       as-is
28   Private Const uniterm_host As String = "localhost"
29   Private Const uniterm_port As Integer = 8123
30
31   ' Authentication information
32   '  NOTE: This information corresponds with the public test server
33   '        at testbox.monetra.com:8665
34   Private Const monetra_user As String = "test_retail:public"
35   Private Const monetra_pass As String = "publ1ct3st"
36
37   Private Function uniterm_path As String
38    Select Case Environment.OSVersion.Platform
39     Case PlatformID.Win32NT, PlatformID.Win32S, _
40          PlatformID.Win32Windows, PlatformID.WinCE
41      Return "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe"
42     Case Else
43      Return "/usr/local/uniterm/bin/uniterm"
44    End Select
45   End Function
46
47
48   '! Function to launch Uniterm from the current process.
49   '  If we don't launch it from the current process, it won't be given
```

```
 50    '  focus! (at least on Windows this is true, until the first
 51    '  manual focus is performed by an end-user)
 52    Private Sub uniterm_launch()
 53     Dim uniterm As New Process()
 54     uniterm.StartInfo.FileName = uniterm_path
 55     uniterm.StartInfo.CreateNoWindow = True
 56
 57     uniterm.Start()
 58
 59     ' Make sure Uniterm is ready before returning,
 60     ' Sleep 1000ms (1s)
 61     System.Threading.Thread.Sleep(1000)
 62    End Sub
 63
 64
 65    '! Function to connect to an endpoint which uses the standard 'monetra'
 66    '  style protocol (so either Monetra itself, or Uniterm)
 67    ' \param[in]  host     Resolvable hostname or IP address to connect to
 68    ' \param[in]  port     Port associated with hostname to establish an SSL
 69    '                      connection to
 70    ' \param[out] errorstr Textual error message if returns null
 71    ' \return Initialized connection class on success. null on failure
 72    Private Function uniterm_connect_host(ByVal host As String, ByVal port _
 73                                    As Integer, ByRef errorstr As String) _
 74                                    As Monetra
 75     ' Initialize the Class
 76     Dim conn As New Monetra
 77
 78     errorstr = ""
 79
 80     ' We always want to use an SSL connection to Monetra and Uniterm
 81     conn.SetSSL(host, port)
 82
 83     ' Do not verify the SSL certificate, Monetra and Uniterm
 84     ' use self-signed certificates by default which cannot be validated.
 85     ' The connection is still encrypted, the endpoint just isn't strictly
 86     ' validated
 87     conn.VerifySSLCert(False)
 88
 89     ' This makes it so TransSend() will block until a response is
 90     ' received from Monetra.  Simplifies the API since we will never
 91     ' have more than one outstanding transaction per connection in
 92     ' this application
 93     conn.SetBlocking(True)
 94
 95     ' Connect!
 96     If Not conn.Connect() Then
 97      errorstr = conn.ConnectionError()
 98      Return Nothing
 99     End If
100
101     Return conn
102    End Function
103
104
105    '! Wrapper function to connect to the Uniterm
106    ' \param[out] errorstr Textual error message if returns null
107    ' \return Initialized connection class on success. null on failure
```

```
108    Private Function uniterm_connect(ByRef errorstr As String) As Monetra
109      Dim conn As Monetra
110      Dim myerror As String = ""
111      conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror)
112      If conn Is Nothing Then
113       errorstr = "Connection to Uniterm Failed: " + myerror
114      End If
115
116      Return conn
117    End Function
118
119    ' Request a transaction from Uniterm as documented in the Uniterm Guide.
120    ' The Key/Value pair params are a combination of the Parameters as
121    ' documented in the Uniterm Guide and the Monetra Client Interface
122    ' Protocol Spec.
123    ' \param[in] uniterm_conn  Initialized connection to the Uniterm
124    '                          as returned by uniterm_connect()
125    ' \param[in] mparams       Array of key/value parameters to send to Uniterm
126    ' \return Hashtable of string key/value pairs from response.  Please refer
127    '         to the Uniterm Guide and Monetra Client Interface Protocol
128    '         specification for the applicable list based on the action being
129    '         performed.  "code" and "u_errorcode" are always guaranteed to
130    '         be returned.
131    Private Function uniterm_sendrequest(ByVal uniterm_conn As Monetra, ByVal _
132                                         mparams As Hashtable) As Hashtable
133      Dim id As Integer
134      Dim response As New Hashtable
135
136      ' Request a new transaction from libmonetra
137      id = uniterm_conn.TransNew()
138
139      ' For each item in the params hashtable, add it to the transaction
140      Dim kv As DictionaryEntry
141      For Each kv In mparams
142       uniterm_conn.TransKeyVal(id, CType(kv.Key, String), _
143                                CType(kv.Value, String))
144      Next kv
145
146      ' Send the request to the Uniterm.  It will not return until a
147      ' response is available, or a disconnect is detected
148      If Not uniterm_conn.TransSend(id) Then
149       ' Disconnect detected, return an appropriate error condition!
150       ' This should really never happen though...
151       response("code")       = "DENY"
152       response("u_errorcode") = "CONN_ERROR"
153       response("verbiage")    = "Connection to Uniterm failed:" _
154                                + uniterm_conn.ConnectionError()
155       Return response
156      End If
157
158      ' Save the response parameters from Uniterm into a
159      ' HashTable as our function prototype states. */
160      Dim keys() As String = uniterm_conn.ResponseKeys(id)
161      Dim i As Integer
162      For i = 0 To keys.Length - 1
163       response(keys(i)) = uniterm_conn.ResponseParam(id, keys(i))
164      Next i
165
```

```
166    ' Free up some memory by purging unneeded data
167    uniterm_conn.DeleteTrans(id)
168
169    Return response
170   End Function
171
172
173   '! Tell Uniterm to shutdown.  Since we start it up,
174   '  we should make sure we turn it off prior to exiting otherwise
175   '  the user will be prompted with an error message stating the
176   '  Uniterm is already running on the next execution
177   '  of this application!
178   ' \param[in] uniterm_conn Initialized connection to Uniterm
179   '                         as returned by uniterm_connect()
180   Private Sub uniterm_shutdown(ByVal uniterm_conn As Monetra)
181    Dim mparams As New Hashtable
182
183    mparams("u_action") = "shutdown"
184    uniterm_sendrequest(uniterm_conn, mparams)
185   End Sub
186
187   '! Main entry point to this application to be executed
188   Public Sub Main()
189    Dim uniterm_conn As Monetra
190    Dim errorstr As String = ""
191    Dim response As Hashtable
192    Dim ticket As String
193
194    ' Step1: Launch Uniterm
195    uniterm_launch()
196    Console.WriteLine("Uniterm Launched")
197
198    ' Step2: Connect to Uniterm
199    uniterm_conn = uniterm_connect(errorstr)
200    If uniterm_conn Is Nothing Then
201     Console.WriteLine("Failure: " + errorstr)
202     Return
203    End If
204    Console.WriteLine("Connected to Uniterm")
205
206    ' Step3: Send a txnrequest to Uniterm
207    Dim mparams As New Hashtable
208    ' Append the parameters for the ticket request as per the Monetra
209    ' Uniterm Guide
210    mparams("username")     = monetra_user
211    mparams("password")     = monetra_pass
212    mparams("u_action")     = "txnrequest"
213    mparams("u_devicetype") = "ingenico_rba"
214    mparams("u_device")     = "HID"
215
216    ' Append the parameters for the transaction that will also get passed
217    ' to Monetra such as the 'action', 'amount', etc. as described in the
218    ' Monetra Client Interface Protocol Specification
219    mparams("action")   = "sale"
220    mparams("amount")   = "12.00"
221    mparams("ordernum") = "123456"
222    mparams("comments") = "u_txnrequest"
223
```

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

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

```
 1 ' UniTerm example program in VBScript
 2 '
 3 ' Depends on the MSXML, and Microsoft Scripting Runtime
 4 '
 5 ' Implemented based on the UniTerm Guide in conjunction with the
 6 ' Monetra Client Interface Protocol Specification
 7 '
 8 ' Please contact support@monetra.com with any questions
 9
10 Option Explicit
11
12 ' Monetra Connectivity Information
13 Dim monetra_user
14 Dim monetra_pass
15
16 ' Uniterm Connectivity Information
17 Dim uniterm_host
18 Dim uniterm_port
19 Dim uniterm_path
20
21
22 '! Function to launch Uniterm from the current process.
23 '  If we don't launch it from the current process, it won't be given
```

```
24  '  focus! (at least on Windows this is true, until the first
25  '  manual focus is performed by an end-user)
26  Sub uniterm_launch()
27   Dim objShell
28   Dim res
29   Set objShell = CreateObject("Wscript.Shell")
30   res = objShell.Run("""" & uniterm_path & """", 10, FALSE)
31
32   ' Make sure Uniterm is ready before returning,
33   ' Sleep 1000ms (1s)
34   WScript.Sleep 1000
35  End Sub
36
37
38  '! Function to POST and XML message to a Monetra-like entity
39  '  (Monetra or the Uniterm) via HTTPS.  It will return
40  '  the key/value pairs from the XML response
41  '\param[in]  host      Host to connect to
42  '\param[in]  port      Port to connect to (via SSL/HTTPS)
43  '\param[in]  xml       String-form XML to post
44  '\param[out] errorstr  If returning False, the error message, typically comms
45  '                      error
46  '\param[out] myresponse  Dictionary of string key/value pairs from the response.
47  '\return True on successful communication, False if communication failed.
48  '        Note: True doesn't mean the transaction itself was successful.
49  Function uniterm_https_post(ByVal host, ByVal port, ByVal xml, ByRef errorstr, _
50                      ByRef myresponse)
51   Dim xmlhttp
52   Dim xmldoc
53
54   Set xmlhttp = CreateObject("MSXML2.ServerXMLHTTP")
55
56   xmlhttp.open           "POST", "https://" & host & ":" & port, False
57   xmlhttp.setOption      2, 13056
58   ' Set Timeouts (in milliseconds)
59   '   DNS: 5s, Connect: 5s, Send: 30s, Receive: 120s
60   xmlhttp.setTimeouts    5000, 5000, 30000, 120000
61   xmlhttp.setRequestHeader "Content-Type", "text/xml"
62
63   On Error Resume Next
64   xmlhttp.send           xml
65
66   If Not Err.Number = 0 Then
67    errorstr = "HTTPS POST Failed to https://" & host & ":" & port & _
68             ": " & Err.Description
69    uniterm_https_post = False
70    Exit Function
71   End If
72
73   Set xmldoc  = CreateObject("Microsoft.XMLDOM")
74
75   xmldoc.async = "false"
76   xmldoc.loadxml(xmlhttp.responseText)
77
78   Dim Trans
79   Set Trans = xmldoc.documentElement.selectSingleNode("Resp").childNodes
80
81   Dim kv
```

```
 82   For Each kv In Trans
 83    myresponse(kv.nodeName) = kv.text
 84   Next
 85
 86   uniterm_https_post = True
 87  End Function
 88
 89
 90  '! Request a transaction from Uniterm as documented in the Uniterm Guide.
 91  '   The Key/Value pair params are a combination of the Parameters as documented
 92  '   the Uniterm Guide and the Monetra Client Interface Protocol Spec.
 93  ' \param[in]  mparams    Dictionary of key/value parameters to send to the
 94  '                        Uniterm
 95  ' \param[out] errorstr   If returning False, the error message, typically comms
 96  '                        error
 97  ' \param[out] myresponse Dictionary of string key/value pairs from response.
 98  '                        Please refer to the Uniterm Guide and Monetra Client
 99  '                        Interface Protocol specification for the applicable
100  '                        list based on the action being performed.  "code" and
101  '                        "u_errorcode" are always guaranteed to be returned.
102  ' \return True on successful communication, False if communication failed.
103  '         Note: True doesn't mean the transaction itself was successful.
104  Function uniterm_sendrequest(ByVal mparams, ByRef errorstr, ByRef myresponse)
105   Dim xml
106
107   xml = "<MonetraTrans><Trans identifier='1'>"
108
109   ' For each item in the params dictionary, add it to the transaction
110   Dim key
111   For Each key In mparams
112    xml = xml & "<" & key & ">" & mparams(key) & "</" & key & ">"
113   Next
114
115   xml = xml & "</Trans></MonetraTrans>"
116
117   uniterm_sendrequest = uniterm_https_post(uniterm_host, uniterm_port, xml, _
118                                            errorstr, myresponse)
119  End Function
120
121
122  '! Tell Uniterm to shutdown.  Since we start it up,
123  '   we should make sure we turn it off prior to exiting otherwise
124  '   the user will be prompted with an error message stating the
125  '   Uniterm is already running on the next execution
126  '   of this application!
127  Sub uniterm_shutdown()
128   Dim myresponse
129   Dim errorstr
130   Dim mparams
131
132   Set mparams = CreateObject("Scripting.Dictionary")
133   mparams("u_action") = "shutdown"
134
135   uniterm_sendrequest mparams, errorstr, myresponse
136   ' No need for error checking in this function as we don't
137   ' care if this fails
138  End Sub
139
```

```
140
141  '! Main entry point to this application to be executed
142
143  ' Uniterm Connectivity Information
144  ' NOTE: this is the default, it is possible to change, but 99%
145  '        of deployments will probably use this Uniterm information
146  '        as-is
147  uniterm_host = "localhost"
148  uniterm_port = 8123
149  uniterm_path = "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe"
150
151  ' Authentication information
152  ' NOTE: This information corresponds with the public test server
153  '        at testbox.monetra.com:8665
154  monetra_user = "test_retail:public"
155  monetra_pass = "publ1ct3st"
156
157
158  Dim errorstr
159  Dim mparams
160  Dim myresp
161  Dim msg
162
163  errorstr = ""
164
165  ' Step1: Launch Uniterm
166  uniterm_launch
167  MsgBox("Uniterm Launched")
168
169
170  ' Step2: Send txnrequest to Uniterm
171
172  Set myresp  = CreateObject("Scripting.Dictionary")
173  Set mparams = CreateObject("Scripting.Dictionary")
174  ' Append the parameters for the txnrequest
175  mparams("username")     = monetra_user
176  mparams("password")     = monetra_pass
177  mparams("u_action")     = "txnrequest"
178  mparams("u_devicetype") = "ingenico_rba"
179  mparams("u_device")     = "HID"
180
181  ' Append the parameters for the transaction that will also get passed
182  ' to Monetra such as the 'action', 'amount', etc. as described in the
183  ' Monetra Client Interface Protocol Specification
184  mparams("action")   = "sale"
185  mparams("amount")   = "12.00"
186  mparams("ordernum") = "123456"
187  mparams("comments") = "u_txnrequest"
188
189  If Not uniterm_sendrequest(mparams, errorstr, myresp) Then
190   MsgBox errorstr
191   WScript.Quit 1
192  End If
193
194  If StrComp(myresp("code"), "AUTH", vbTextCompare) <> 0 Then
195   msg = "Transaction failed." & vbNewLine
196  Else
197   msg = "Transaction SUCCESSFUL!" & vbNewLine
```

```
198  End If
199
200  ' Print out all the response key/value pairs ...
201  Dim key
202  For Each key In myresp
203   msg = msg & "  " & key & " = " & myresp(key) & vbNewLine
204  Next
205
206  MsgBox (msg)
207
208
209  ' NOTE: No real reason to exit here ... we could just keep running
210  '        Step 2 all day long.  No reason to keep starting/stopping the
211  '        Uniterm.
212
213  ' Step3: Cleanup
214  uniterm_shutdown
215
216
217
218
219
```

E.7 Microsoft Visual Basic 6 using libmonetra

```
 1  Attribute VB_Name = "Module1"
 2  ' UniTerm example program in VB6
 3  '
 4  ' Depends on the libmonetra C# .Net native API (DLL) (has COM hooks)
 5  '
 6  ' Must add reference to libmonetra and Microsoft Scripting Runtime
 7  '
 8  ' Implemented based on the UniTerm Guide in conjunction with the
 9  ' Monetra Client Interface Protocol Specification
10  '
11  ' Please contact support@monetra.com with any questions
12
13  Option Explicit
14
15  ' MonetraInformation
16  Dim monetra_user As String
17  Dim monetra_pass As String
18
19  ' Uniterm Connectivity Information
20  Dim uniterm_host As String
21  Dim uniterm_port As Integer
22  Dim uniterm_path As String
23
24  Private Declare Sub Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)
25
26  '! Function to launch Uniterm from the current process.
27  '  If we don't launch it from the current process, it won't be given
28  '  focus! (at least on Windows this is true, until the first
29  '  manual focus is performed by an end-user)
30  Sub uniterm_launch()
31   Dim id As Double
32   id = Shell("""" & uniterm_path & """", vbNormalFocus)
```

```
33
34   ' Make sure Uniterm is ready before returning,
35   ' Sleep 1000ms (1s)
36   Sleep (1000)
37 End Sub
38
39
40 '! Function to connect to an endpoint which uses the standard 'monetra'
41 ' style protocol (so either Monetra itself, or Uniterm)
42 ' \param[in]  host     Resolvable hostname or IP address to connect to
43 ' \param[in]  port     Port associated with hostname to establish an SSL
44 '         connection to
45 ' \param[out] errorstr Textual error message if returns null
46 ' \return Initialized connection class on success. null on failure
47 Function uniterm_connect_host(ByVal host As String, ByVal port As Integer, _
48     ByRef errorstr As String) As IMonetra
49   ' Initialize the Class
50   Dim conn As IMonetra
51   Set conn = New Monetra
52
53   errorstr = ""
54
55   ' We always want to use an SSL connection to Monetra and Uniterm
56   conn.SetSSL host, port
57
58   ' Do not verify the SSL certificate, Monetra and Uniterm
59   ' use self-signed certificates by default which cannot be validated.
60   ' The connection is still encrypted, the endpoint just isn't strictly
61   ' validated
62   conn.VerifySSLCert False
63
64   ' This makes it so TransSend() will block until a response is
65   ' received from Monetra.  Simplifies the API since we will never
66   ' have more than one outstanding transaction per connection in
67   ' this application
68   conn.SetBlocking True
69
70   ' Connect!
71   If Not conn.Connect() Then
72    errorstr = conn.ConnectionError()
73    Set uniterm_connect_host = Nothing
74    Exit Function
75   End If
76
77   Set uniterm_connect_host = conn
78 End Function
79
80
81 '! Wrapper function to connect to Uniterm
82 ' \param[out] errorstr Textual error message if returns null
83 ' \return Initialized connection class on success. null on failure
84 Function uniterm_connect(ByRef errorstr As String) As IMonetra
85   Dim conn As IMonetra
86   Dim myerror As String
87
88   myerror = ""
89   Set conn = uniterm_connect_host(uniterm_host, uniterm_port, myerror)
90   If conn Is Nothing Then
```

```
 91    errorstr = "Connection to Uniterm Failed: " & myerror
 92   End If
 93   Set uniterm_connect = conn
 94  End Function
 95
 96
 97  ' Request a transaction from Uniterm as documented in the UniTerm
 98  ' Guide.  The Key/Value pair params are a combination of the Parameters as
 99  ' Uniterm Guide and the Monetra Client Interface Protocol Spec.
100  ' \param[in] uniterm_conn  Initialized connection to Unitermas returned by
101  '                          connect_to_uniterm()
102  ' \param[in] mparams       Dictionary of key/value parameters to send to
103  '                          Uniterm
104  ' \return Dictionary of string key/value pairs from response.  Please refer
105  '  to the Uniterm Guide and Monetra Client Interface Protocol
106  '  specification for the applicable list based on the action being
107  '  performed.  "code" and "u_errorcode" are always guaranteed to
108  '  be returned.
109  Function uniterm_sendrequest(ByVal uniterm_conn As IMonetra, _
110      ByVal mparams As Dictionary) _
111      As Dictionary
112   Dim id As Integer
113   Dim myresponse As New Dictionary
114
115   ' Request a new transaction from libmonetra
116   id = uniterm_conn.TransNew()
117
118   ' For each item in the params dictionary, add it to the transaction
119   Dim key
120   For Each key In mparams
121    uniterm_conn.TransKeyVal id, key, mparams(key)
122   Next key
123
124   ' Send the request to the Uniterm.  It will not return until a
125   ' response is available, or a disconnect is detected
126   If Not uniterm_conn.TransSend(id) Then
127    ' Disconnect detected, return an appropriate error condition!
128    ' This should really never happen though...
129    myresponse("code") = "DENY"
130    myresponse("u_errorcode") = "CONN_ERROR"
131    myresponse("verbiage") = "Connection to Uniterm failed: " _
132        & uniterm_conn.ConnectionError()
133    Set uniterm_sendrequest = myresponse
134    Exit Function
135   End If
136
137   ' Save the response parameters from the Uniterm into a HashTable
138   ' as our function prototype states.
139   Dim keys() As String
140   keys = uniterm_conn.ResponseKeys(id)
141   Dim i As Integer
142   For i = LBound(keys) To UBound(keys)
143    myresponse(keys(i)) = uniterm_conn.ResponseParam(id, keys(i))
144   Next i
145
146   ' Free up some memory by purging unneeded data
147   uniterm_conn.DeleteTrans (id)
148
```

```
149   Set uniterm_sendrequest = myresponse
150 End Function
151
152
153 '! Tell Uniterm to shutdown.  Since we start it up, we should make sure we
154 '  turn it off prior to exiting otherwise the user will be prompted with an
155 '  error message stating Uniterm is already running on the next execution
156 '  of this application!
157 ' \param[in] uniterm_conn Initialized connection to Uniterm as returned by
158 '              connect_to_uniterm()
159 Sub uniterm_shutdown(ByVal uniterm_conn As IMonetra)
160  Dim mparams As New Dictionary
161
162  mparams("u_action") = "shutdown"
163  uniterm_sendrequest uniterm_conn, mparams
164 End Sub
165
166
167 '! Main entry point to this application to be executed
168 Sub Main()
169  ' Uniterm Connectivity Information
170  ' NOTE: this is the default, it is possible to change, but 99%
171  '       of deployments will probably use this Uniterm information
172  '       as-is
173  uniterm_host = "localhost"
174  uniterm_port = 8123
175  uniterm_path = "C:\\Program Files\\Main Street Softworks\\UniTerm\\uniterm.exe"
176
177  ' Authentication information
178  ' NOTE: This information corresponds with the public test server
179  '       at testbox.monetra.com:8665
180  monetra_user = "test_retail:public"
181  monetra_pass = "publ1ct3st"
182
183  Dim uniterm_conn As IMonetra
184  Dim errorstr As String
185  Dim myresp As Dictionary
186  Dim msg As String
187
188  errorstr = ""
189
190  ' Step1: Launch Uniterm
191  uniterm_launch
192  MsgBox ("Uniterm Launched")
193
194  ' Step2: Connect to Uniterm
195  Set uniterm_conn = uniterm_connect(errorstr)
196  If uniterm_conn Is Nothing Then
197   MsgBox ("Failure: " & errorstr)
198   Exit Sub
199  End If
200
201  MsgBox ("Connected to the Uniterm")
202
203  ' Step3: Send a txnrequest to Uniterm
204  Dim mparams As New Dictionary
205  ' Append the parameters for the ticket request as per the Uniterm Guide
206  mparams("username")     = monetra_user
```

```
207   mparams("password")     = monetra_pass
208   mparams("u_action")     = "txnrequest"
209   mparams("u_devicetype") = "ingenico_rba"
210   mparams("u_device")     = "HID"
211
212   ' Append the parameters for the transaction that will also get passed
213   ' to Monetra such as the 'action', 'amount', etc. as described in the
214   ' Monetra Client Interface Protocol Specification
215   mparams("action")       = "sale"
216   mparams("amount")       = "12.00"
217   mparams("ordernum")     = "123456"
218   mparams("comments")     = "u_txnrequest"
219
220   Set myresp = uniterm_sendrequest(uniterm_conn, mparams)
221   If StrComp(myresp("code"), "AUTH", vbTextCompare) <> 0 Then
222    msg = "Transaction failed." & vbNewLine
223   Else
224    msg = "Transaction SUCCESSFUL!" & vbNewLine
225   End If
226
227   ' Print out all the response key/value pairs ...
228   Dim key
229   For Each key In myresp
230    msg = msg & "  " & key & " = " & myresp(key) & vbNewLine
231   Next key
232   MsgBox (msg)
233
234   ' NOTE: No real reason to exit here ... we could just keep running
235   '       Step 3 all day long as long as you keep the uniterm_conn handle.
236   '       No reason to keep disconnecting and reconnecting, or
237   '       starting/stopping Uniterm.
238
239   ' Step4: Cleanup
240   uniterm_shutdown uniterm_conn
241
242   ' Connections will be automatically closed when the uniterm_conn initialized
243   ' class is cleaned up by the destructor/garbage collector
244  End Sub
245
246
247
```

F PCI Security and Implementation

The below details the various security and PCI requirements and how deployments may be impacted. Integrators and distributors should read this section prior to any production deployments. UniTerm is designed to be compliant with all PCI PA-DSS requirements and cannot be configured to be non-compliant.

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

Note: Please use this section along with the official PCI DSS v3.2 specification available at https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf

TOPIC	DISCUSSION
Delete sensitive authentication data stored by previous payment application versions.	UniTerm has never stored any sensitive authentication data in a non-secured or non-approved manner.
Delete any sensitive authentication data (pre-authorization) gathered as a result of troubleshooting the payment application.	UniTerm does not have the ability to store sensitive authentication data for troubleshooting purposes.
Securely delete cardholder data after customer-defined retention period.	UniTerm never stores cardholder data after authorization and prior to authorization is stored as per Section 7.1, and Section 7.3.
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. There is no ability to configure UniTerm to return the PAN.
Render PAN unreadable anywhere it is stored (including data on portable digital media, backup media, and in logs).	Sensitive cardholder data is always unreadable as per Section 7.1 and Section 7.2, this is always true, there is no configuration option that controls this behavior.
Protect keys used to secure cardholder data against disclosure and misuse.	UniTerm does not have the keys to decrypt any stored cardholder data as per Section 7.1
Implement key-management processes and procedures for cryptographic keys used for encryption of cardholder data.	Only ephemeral keys are used to store cardholder data, and those are protected by RSA Public keys rotated ever 10-12 days as per Section 7.1. There is no configuration option to control this behavior, it is always implemented as documented.
Implement secure key-management functions.	UniTerm requests RSA public keys from Monetra, Monetra handles management of the RSA private keys in its

---

	validated CardShield subsystem. See Section 7.1, there is no configuration option to control this behavior, it is always implemented as documented.
Provide a mechanism to render irretrievable cryptographic key material or cryptograms stored by the payment application.	UniTerm never has the keys to decrypt sensitive cardholder data as per Section 7.1. There is no configuration option to control this behavior, it is always implemented as documented.
Use unique user IDs and secure authentication for administrative access and access to cardholder data.	UniTerm does not provide or facilitate administrative access, all access controls are provided by the remote Monetra instance.
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. UniTerm utilizes the validated Monetra authentication subsystem for access to UniTerm and its connected devices in conjunction with the policies as described in Section 7.1 All access controls to cardholder data are provided by the remote Monetra instance. Note: It is the integrator's responsibility to ensure unique user names and secure authentication are used to access any PCs, servers, and databases with payment applications and/or cardholder data. This is requirement is external to UniTerm and not something that can be provided by UniTerm on behalf of the merchant or integrator.
Implement automated audit trails.	Audit trails are provided by both logging on the Monetra side of all end-user transactions as well as UniTerm's built-in logging mechanisms as described in Section 7.2
Facilitate centralized logging.	Centralized logging is facilitated by the syslog subsystem provided by UniTerm as described in Section 7.2
Implement and communicate application versioning methodology.	Please see the Versioning section.
Securely implement wireless technology.	UniTerm is not designed facilitate the use of wireless technologies, however merchants may deploy UniTerm in environments where wireless technologies are in use. UniTerm always uses secure communications channels protected by TLS v1.2 or higher, which are designed for use in public/untrusted networks, to protect sensitive cardholder data while in transit so requiring additional security mechanisms on any wireless technology in use is not a requirement as per PCI, however it is always a recommended best practice.

Secure transmissions of cardholder data over wireless networks.	UniTerm is not designed facilitate the use of wireless technologies, however merchants may deploy UniTerm in environments where wireless technologies are in use. UniTerm always uses secure communications channels protected by TLS v1.2 or higher, which are designed for use in public/untrusted networks, to protect sensitive cardholder data while in transit so requiring additional security mechanisms on any wireless technology in use is not a requirement as per PCI, however it is always a recommended best practice.
Provide instructions for secure use of wireless technology.	UniTerm is not designed facilitate the use of wireless technologies, however merchants may deploy UniTerm in environments where wireless technologies are in use. UniTerm always uses secure communications channels protected by TLS v1.2 or higher, which are designed for use in public/untrusted networks, to protect sensitive cardholder data while in transit so requiring additional security mechanisms on any wireless technology in use is not a requirement as per PCI, however it is always a recommended best practice. Integrators should ensure they secure any wireless technologies in use are in compliance with the requirements in PA-DSS Requirement 6.3 for other applications that may not deploy UniTerm's secure practices.
Use only necessary and secure services, protocols, components and dependent software and hardware, including those provided by third parties.	UniTerm communicates only via TLS using proprietary protocols to a Monetra server (hosted or customer-owned) across an intranet or the Internet. A customer may choose to deploy UniTerm with one or more hardware card entry devices or terminals directly attached to the System via USB, Serial, BlueTooth, or Ethernet

It is the integrator's responsibility to ensure only necessary and secure protocols, services, etc., are used on the system. |
Store cardholder data only on servers not connected to the Internet	Cardholder data is only stored within UniTerm prior to authorization with no ability to retrieve it in plain-text form as per Section 7.1
Implement two-factor authentication for all remote access to payment application that originates from outside the customer environment.	UniTerm does not facilitate remote access, therefore integrators or merchants choosing to provide an external means of remote access must ensure that all remote access originating from outside the customer's network to a payment application must use two-factor authentication. Two-factor means two separate types of authentication, a username and password is one factor, a second password would not be considered a second factor, it should instead be something external such as a token.
Securely deliver remote payment application updates.	Integrators or merchants 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.	Monetra Technologies will never reach out to a remote customer network. If an integrator or merchant chooses to support remote access for management they must do so in compliance with PCI DSS requirements, specifically section 8.
Secure transmissions of cardholder data over public networks.	UniTerm communicates only via TLS to Monetra using proprietary protocols. UniTerm communicates using the PCI DSS required protocols and cipher suites automatically (`TLSv1.2 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`). No configuration is required of UniTerm to comply, nor is it possible to reconfigure UniTerm to a less secure protocol or cipher suite. Future versions may implement new protocols and cipher suites as they become available. UniTerm, by default performs full validation of the remote Monetra's certificate. This must not be disabled if connecting to Monetra over the public internet as per Section 4.2.1's' `ssl_cert_validate`.
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.