PrismVend Web Vending API

Document number	PR-D2-1112
Revision	5.20
Authors	Trevor Davel, Chris Adlington, Candice Moore
Date	May 2023
Synopsis	Describes the PrismVend Web Vending Service and API.

1. Contents

2. Installing & Using the Web Service

2.1 Installation

Refer to PR-D2-1098 “PrismVend User Guide”.

2.2 Starting & Stopping the service

The *Prism TsmWeb* service may be controlled via the Windows Service Control Panel.

There are various ways to access the Service Control Panel:

- Computer Manager → Services and Applications → Services
- Administrative Tools → Services
- Start → Run → services.msc

Refer to PR-D2-1098 “PrismVend User Guide”.

2.3 Configuration

Refer to PR-D2-1098 “PrismVend User Guide”.

2.4 Accessing web services

Use your web browser to access the URL http://localhost:80/ to view the human-consumable services provided by the software.

- If you have changed the service configuration you may need to specify a different host or port in the URL.
- Services for automation, as documented in 3. Application Programming Interface (API), cannot be accessed via a web browser. You must access them programmatically or using a Debugging tool (section 3.6).

2.5 Security procedures in a production environment

Hardware Security Modules and cryptographic services should be operated with due regard to security procedures designed to protect keys and prevent misuse of the services.

- Security procedures are outside the scope of this document.
- Networks and network-accessible services should be secured against unintended access, for example by means of firewalls.

2.6 Troubleshooting

Problems starting the service:

- Check the System log in the Windows Event Viewer (accessible via Computer Manager or Administrative Tools) for messages from the Service Control Manager.
- Check the **web service log**: `%INSTALL_DIR%\tap.tcl.err`

Problems accessing the service (networking & TCP/IP connectivity):

- Use the Service Control Panel or `SC.EXE` utility to check that the service is running.
- Check the web service logs for errors; the log will also indicate the TCP/IP address and port at which the service is listening.
- Check that your system is in fact listening for connections on the expected TCP/IP address and port by using the command `netstat.exe -an`
- Check the firewall settings in Control Panel → Windows Firewall.

- If the web service is running on another computer, use the `ping.exe`, `telnet.exe` and `tracert.exe` utilities to ensure that basic TCP/IP communication exists between the computers.

  `ping.exe` and `tracert.exe` are built for network diagnostics but use the ICMP protocol and may be fooled by firewalls. If you can use `telnet.exe` to connect to the web server's address and port then the network and service are behaving correctly.

Problems accessing the Hardware Security Module (HSM) or performing cryptographic operations:

- Use the StsClosedVending UI to diagnose the problem.

Problems using the web services:

- Refer to 3.6 Debugging.

2.7 Upgrading the software

Upgrades follow the same procedure as Installation. You must install an upgrade into the same folder as the original installation (the setup application will detect the correct folder).

2.8 Uninstalling the software

An Uninstall shortcut is provided in the `Start` menu under `All Programs` → `Prism` → `TsmWeb`. You can also uninstall the software via Control Panel → Add or Remove Programs.

- Some files – such as configuration and logs – will not be removed by the uninstall application. You can manually remove them from the `%INSTALL_DIR%` if required.

3. Application Programming Interface (API)

3.1 Overview

Prism's web service APIs are designed to follow the REST[1] model:

- Where possible the services are stateless and resource oriented.
- Some services perform computations rather than resource manipulation and are exposed as REST-like remote procedure calls. Such services may only be accessed via the HTTP POST method.
- The HTTP GET, HEAD, PUT and DELETE methods are always idempotent (side-effect free). POST is seldom (if ever) idempotent.
- Some services support queries[2] (parameters). The query string is appended to the URL for a GET request, or included in the request body for POST. The query string is built and encoded according to the HTTP specification.
    - XML, SOAP, or JSON encoded requests are not supported.
- The resource or output obtained from a service may be retrieved in different formats or *representations*. The client indicates the desired representation in the URL.
    - Web services meant for human consumption are generally only available in HTML format.
    - Web services meant for automation are generally available as INI, TSV or XML.
- The result of processing (success or failure) is indicated by HTTP Status Codes.
    - A 200 "OK" status indicates that the service executed successfully, and the body of the response will contain the requested resource or service output.
    - All other codes have meanings as defined in the HTTP specification.

3.2 Terms, definitions and abbreviations

The table below describes the abbreviations used throughout this document.

BDT	BaseDate
CC	CountryCode
CRC	CyclicRedundancyCode
DAC	DeviceAuthenticationCode
DOE	DateOfExpiry
DRN	DecoderReferenceNumber (meter number)
DSN	DecoderSerialNumber
DUTK	DecoderUniqueTransferKey
EA	EncryptionAlgorithm
FAC	FirmwareAuthenticationCode
IAIN	IndividualAccountIdentificationNumber

[1] Representational State Transfer: http://en.wikipedia.org/wiki/Representational_State_Transfer.

[2] See http://en.wikipedia.org/wiki/Query_string.

ID	Identification; Identifier
IIN	IssuerIdentificationNumber
ISO	International Standards Organisation
ISO	BIN Replaced by IIN
KCT	KeyChangeToken
KEK	KeyExchangeKey
KEN	KeyExpiryNumber
KLF	KeyLoadFile
KRN	KeyRevisionNumber
KT	KeyType
LRC	LongitudinalRedundancyCheck
Mfr	Manufacturer
MPL	MaximumPowerLimit
MPPUL	MaximumPhasePowerUnbalanceLimit
PAN	PrimaryAccountNumber
POS	PointOfSale
RND	RandomNumber
RO	Roll over
SG	SupplyGroup
SGC	SupplyGroupCode
SHA	Secure Hash Algorithm
STA	Standard Transfer Algorithm
STS	Standard Transfer Specification
STSA	Standard Transfer Specification Association
TCDU	TokenCarrierDataUnit
TCT	TokenCarrierType
TDEA	Triple Data Encryption Algorithm
TI	TariffIndex
TID	TokenIdentifier
UC	UtilityCode
VCDK	VendingCommonKey
VDDK	VendingDefaultKey
VK	VendingKey
VUDK	VendingUniqueKey

3.3 Access control and authentication

The current generation web services are provided as an integration technology only – much like a DLL, COM object, or Prism's Conductor service – and do not provide access control or authentication mechanisms.

The services are meant to run on a secured network or on localhost (127.0.0.1), and should never be directly accessible to end-users.

3.4 Status codes

The status codes that may be returned by a Prism web service are consistent with the HTTP specification[3]. The table below contains commonly encountered codes and information on the probably cause of the code.

In the case of client errors (codes 4xx) or server errors (codes 5xx) the response body will include more specific information on the cause or nature of the error.

Status Code	Meaning	Probable cause
200	OK	The operation completed successfully.
302	Found *also known as* *Moved Temporarily*	The client should access the resource using a different URL (the correct URL will be in the `Location:` header).
400	Bad Request	The resource or service was found, but the request is inappropriate. Common causes include: ▪ Missing or extra query parameters ▪ Invalid or out-of-range parameter values ▪ Trailing path information in the URL
404	Not Found	No document, resource or service was found at the requested URL.
405	Method Not Allowed	The resource or service was found, but does not support the requested method of access (HTTP GET, POST, etc.)
415	Unsupported Media Type	The requested representation is not recognised, or is not compatible with the resource or service.
500	Internal Server Error	A runtime exception has occurred and the operation could not complete.
501	Not Implemented	The requested service is not implemented in this version of the software.
503	Service Unavailable	The requested resource or service is temporarily unavailable. This is usually caused by a dependency such as a database, network file system, or coprocessor being offline.

3.5 Output formats (representations)

Web services meant for automation may produce their outputs in one of several representations. Most services support the following representations:

3.5.1 INI (default)

INI format consists of lines of `key=value` pairs, like a Windows INI file.

[3] HTTP status codes: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

- The `key` is strictly alphanumeric.
- The value may contain any character that may be represented in the *charset* of the HTTP response except the line delimiter.
- The line delimiter is ASCII character LF = 10 (\n).

Example:

```
token=67701832116585202773
tokenHex=3AB8D399C40318455
txCredit=974
```

3.5.2 TSV

Tab-Separated Value format is a list of alternating keys and values, delimited by the ASCII character TAB = 8 (\t).

- The keys are strictly alphanumeric.
- The values may contain any character that may be represented in the *charset* of the HTTP response except the TAB delimiter and the ASCII character LF = 10 (\n).
- A TSV response is contained within a single line; this format is thus not suitable for large responses.

Example (the TAB delimiter is indicated by →)

```
token→21701356005869838267→tokenHex→12D2AB44F02295FBB→txCredit→973
```

3.5.3 XML

The XML format is specified[4] by the W3C. Responses generated by Prism's web services use a subset of XML 1.0.

- No processing instructions, comments, or CDATA sections are used.
- Namespaces are not used, and elements (tag) may not have attributes.
- Empty tags are not used – an empty value will have an open and close tag.
- No DTD or schema is used – the meaning of the XML data is defined by the web service API specification.
- The entire response is contained with a `<response>` element.
- Each output field will be represented as an element within the `<response>`. The element name (tag) indicates the name of the output field. The content of the element is the value of the field.
- Encoding is not relevant to this API: all output values are represented as alphanumeric characters and will not require encoding to be included in an XML document.

Example:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<response>
  <token>60977247870879488672</token>
  <tokenHex>34E3AB3DE04B75AA0</tokenHex>
  <txCredit>972</txCredit>
</response>
```

3.6 Debugging

It is common to encounter problems during the development and integration of software components. In the Web Services environment these problems often relate to invalid data

[4] Extensible Markup Language (XML): http://www.w3.org/XML/Core/.

encodings, caching, redirection or bad URLs. Such integration problems may be difficult to debug with a traditional IDE.

Prism has found the following tools, techniques and information sources to be helpful:

- Check the web service logs (see also 2.6 Troubleshooting).

  `%INSTALL_DIR%\tap.tcl.err`

- For networking and TCP/IP connectivity problems, refer to 2.6 Troubleshooting.
- For HSM and cryptographic subsystem problems, refer to 2.6 Troubleshooting.
- For web service testing and integration issues we recommend the cURL utility, an Open Source tool available from http://curl.haxx.se/. `curl` runs on many platforms including Windows, Linux, Mac OS/X, Solaris and HP-UX.

  `curl` allows you to access web services from the command line, controlling the HTTP method and headers, URL, and request parameters. The full HTTP response including the resource code, headers and body can be displayed to give full visibility of the web service protocol.

3.7 Tariff File Format

The tariff database can be updated by importing a tariff file. A tariff file is a text file containing records for each supply group code given a particular tariff index, subclass and active date. Each record consists of comma separated values where the first value of the record is a prefix describing the tariff record structure. Currently only “Tariff1” is supported by the system.

All fields are in ASCII for the length indicated. Representation is defined as follows:

a alphabetic

n numeric

an alpha-numeric

ah alpha-hex

ans alpha-numeric, special

Fields:

Prefix	an
Supply group code	n(6)
Tariff index	n(2)
Subclass	n(2)
Date active	n(10)
Tariff	n(1-*)

Field Descriptions

- **Prefix**

  The tariff record type for this entry. Currently only “Tariff1” is supported.

- **Supply group code**

  The supply group code the tariff applies to

- **Tariff index**

  The tariff index the tariff applies to for the given supply group. A tariff index less than 10 must be space padded with a zero

- **Subclass**

  The subclass the tariff applies to given the supply group and tariff index. E.g. water or electricity. Left padded with zero for a subclass less than 10

- **Date Active**

  The date from which this tariff value becomes active for that supply group.

- **Tariff**

  The tariff value in cents per 100Wh

Example:

```
# Comment line
Tariff1,123456,01,00,1382004571,50
Tariff1,123456,01,01,1330000000,90
Tariff1,888888,02,00,1382004571,30
```

4. PrismVend API

The PrismVend API provides services for STS token vending.

4.1 API services

- Vend Credit Token
- Vend Meter-Specific Engineering Token
- Query Transaction Counter

4.2 Legacy API services

Legacy API services are provided for compatibility with Prism's STSClosedVending product (v3.x). Do not use these services for new development.

- Calculate Watt-Hours (LEGACY)

4.3 Vend Credit Token

Generates an IEC 62055-41 Class 0 “Credit Transfer Token” to transfer credit for any service type (determined by subclass) to a given meter.

URL	`http://host/stsvend/VendCredit.`*`format`*
Formats	tsv *(default)*, ini, xml
HTTP method(s)	POST

Input Parameters (Required)	**meterId** (required)
	Identifies the meter that will consume the token.
	Format: A printable ASCII string in any of the following formats:
	• For blind vending (to unregistered meters):
	  ○ 35 digit IDRecord (MeterPAN, DOE, TCT, EA, SGC, TI, KRN)
	  ○ Record2 (MagStripe) format with or without sentinels and LRC
	• For vending to registered meters:
	  ○ 11 digit DecoderReferenceNumber (DRN) (containing manufacturer code, device SN, Luhn/DRNCheckDigit)
	  ○ 16 digit PAN (as for 17 digit PAN but excludes first digit of the IIN)
	  ○ 17 digit PAN (as for 18 digit PAN but excludes trailing Luhn/PANCheckDigit)
	  ○ 18 digit PAN (IIN, country code, DSN, Luhn/PANCheckDigit)
	The Security Module in the vending server must have the Vending Key for the Supply Group Code on which the meter is registered.

	subclass (required) Gives the subclass of the STS Class 0 token, which determines the resource represented by the token (electricity, water, gas, ...). Format: A decimal integer in the range -1 to 15. If -1 is specified, the CDU will do a database lookup to determine the resource type of the meter that has been registered on the system. The following values are accepted: • -1 = Auto • 0 = Electricity • 1 = Water • 2-15 = Reserved
	value (required) The desired currency value of the token, net of fees, expressed in “cents” (0.01 base currency units). To vend R50.01 you must supply value=5001. This value will be converted into a number of resource units at the ruling Tariff (for the meter's installed location and configuration), then encoded into an STS TransferAmount per the STS standard. The encoding places an upper bound of 18,201,624 on the number of units. Format: A floating point number (up to 3 digits precision); must be greater than or equal to 0 for subclasses 0-3. v3.29.0: value is not net of fees, specification to be revisited.

	messageId (optional, default empty) A unique transaction identifier. MessageId may be used by clients to safely resubmit a transaction when the outcome of a previous attempted submission is unknown. Other uses of messageId include finding transactions for reprinting, reporting, diagnostics or fraud investigation. A messageId is optional but strongly recommended. If omitted (or given as an empty string) then a unique messageId will be assigned by the server. Any other value must be unique per transaction; the transaction will be rejected as a duplicate if an identifier is reused. We recommend that the messageId be constructed with the originating client or terminal ID as a prefix, and a transaction counter or timestamp as the suffix. You may want to include the username or the authenticated operator. We also recommend that the suffix has an ASCII sort order equivalent to the transaction order. For example we could use a POS identifier, an employee number, and an ISO 8601 point-in-time: "POS.23.4-emp0139-20130818T154022Z". Format: A printable ASCII string string containing only alphanumeric characters or underscore '_', hyphen '-', period '.' or comma ','; with a maximum length of 40 characters (equivalently: the value must match the Perl-Compatible regular expression '^[a-zA-Z0-9_\-.,]{0,40}$').
	vendType (optional, default 1) Indicates whether the operation is a Trial vend (0) or a Prepayment vend (1). Format: A decimal integer greater than 0.
Status codes	200, 400, 402, 405, 415, 500, 503
Output	The output will include the following fields: • **idRecord** 35 digit IDRecord per IEC 62055-41 (contains MeterPAN, DOE, TCT, EA, SGC, TI, KRN). • **subclass** As for input. • **description** A human-readable description of the token type. • **vendTimeUnix** The date & time of the vend given as a Unix timestamp (seconds elapsed since the Epoch, 1970-01-01 00:00) expressed as a decimal integer. • **unitsActual** The actual number of transfer units issued, as a floating point number. The scale/measure is given by **unitName**. • **unitName** The name of the transfer unit for the subclass, as a printable ASCII string (e.g. “kWh”, “kL”). • **valueActual** The monetary value of the credit represented by the token (net of fees), given in 0.01 base currency units, expressed as a floating point number (up to 3 decimal places precision). o This field typically equal to the input **value**, but may be slightly larger due to rounding (the STS standard requires all rounding to be in the customer’s favour). o $\text{valueActual} = \text{tariff} \times \text{unitsActual} \times 100$ • **Tariff** The applicable tariff (base currency units per resource unit [given by **unitName**]), expressed as a floating point number. • **tokenDec** The generated token as 20 decimal digits, suitable for

	printing on a POS slip for entry into a meter via a keypad. • **tokenHex** The generated token as 17 hexadecimal characters, suitable for transfer to a magnetic token carrier. • **tillslipSv** (since v5.15) Detail of how the input **value** was allocated, as a pipe ('|') separated list of description and amount. v3.29.0: valueActual and tariff changed from "decimal integer" to "floating point" v4.23.8: unitsActual changed from "decimal integer" to "floating point".

Example of use #1 – XML response

Request

```
curl http://localhost:8080/stsvend/VendCredit.xml -d subclass=0 -d
meterId=600727000000000009===0207123456011 -d value=50
```

Response

```
<?xml version='1.0' encoding='utf-8'?>
<response><idRecord>600727000000000090000207123456011</idRecord>
<tariff>11.0</tariff>
<subclass>0</subclass>
<description>Sale - Credit:Electricity</description>
<vendTimeUnix>1378384620</vendTimeUnix>
<unitsActual>0.5</unitsActual>
<unitName>kWh</unitName>
<valueActual>55</valueActual>
<tokenHex>1111111111111111</tokenHex>
<tokenDec>22222222222222222</tokenDec>
</response>
```

Example of use #2 – INI response (with HTTP headers shown)

Request

```
curl -i http://localhost:8080/stsvend/VendCredit.ini -d subclass=0 -d
meterId=600727000000000009===0207123456011 -d value=100
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 05 Sep 2013 12:38:06 GMT
Server: Prism WSHOST
set-cookie: nsssession=52287b2e.fK S9YptbHbIhUB8zz5jEQ; Path=/
content-length: 233
content-type: text/plain; charset=utf-8
cache-control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-
check=0
expires: Sun, 01 Jul 2005 00:00:00 GMT
pragma: no-cache
vary: Accept-Encoding

idRecord=600727000000000090000207123456011
tariff=11.0
subclass=0
description=Sale - Credit:Electricity
vendTimeUnix=1378384860
unitsActual=1.0
unitName=kWh
valueActual=110
tokenHex=1111111111111111
tokenDec=22222222222222222
```

4.4 Vend Meter-Specific Engineering Token

Generates an IEC 62055-41 Class 2 “Meter-specific Engineering Token” to issue a management instruction (determined by subclass) to a given meter.

<table>
<tr><td>URL</td><td>http://host/stsvend/VendMse.format</td></tr>
<tr><td>Formats</td><td>tsv (default), ini, xml</td></tr>
<tr><td>HTTP method(s)</td><td>POST</td></tr>
<tr><td>Input Parameters (Required)</td><td>meterId (required) As for Vend Credit Token.</td></tr>
<tr><td></td><td>Subclass (required)
Gives the subclass of the STS Class 2 token, which determines the management instruction represented by the token.
<table>
<tr><th>Sub class</th><th>Management function</th><th>Supp range</th></tr>
<tr><td>0</td><td>SetMaximumPowerLimit</td><td>0 to 18,201,624</td></tr>
<tr><td>1</td><td>ClearCredit</td><td>0 to 65535</td></tr>
<tr><td>5</td><td>ClearTamperCondition</td><td>0</td></tr>
<tr><td>6</td><td>SetMaximumPhasePowerUnbalanceLimit</td><td>0 to 18,201,624</td></tr>
<tr><td>7</td><td>SetWaterMeterFactor</td><td>0 to 65535</td></tr>
</table>
Format: A decimal integer in the range 0 to 15 (excluding 3 or 4, which are reserved for Key Change).</td></tr>
<tr><td></td><td>Supp (optional, default 0)
Supplementary data value to be encoded into the token. The interpretation of this value is subclass-specific; refer to IEC 62055-41.
Format: A decimal integer; the range is subclass-specific.</td></tr>
<tr><td></td><td>messageId (optional, default empty) As for Vend Credit Token.</td></tr>
<tr><td>Status codes</td><td>200, 400, 402, 405, 415, 500, 503</td></tr>
<tr><td>Output</td><td>The output will include the following fields:
idRecord As for Vend Credit Token.
subclass As for input.
description A human-readable description of the token type.
vendTimeUnix As for Vend Credit Token.
unitsActual The actual number of transfer units issued, as a decimal integer.
unitName The name of the transfer unit for the subclass, as a printable ASCII string (e.g. subclass 0 units are 100Wh so unitName is “hWh”).
tokenDec The generated token as 20 decimal digits, suitable for printing on a POS slip for entry into a meter via a keypad.
tokenHex The generated token as 17 hexadecimal characters, suitable for transfer to a magnetic token carrier.
</td></tr>
</table>

4.5 Query Transaction Counter

Returns the number of transactions left in the module.

URL	`http://host/stsvend/QueryTx.format`
Formats	tsv, ini *(default)*, xml
HTTP method(s)	GET
Input Parameters (Required)	*None*
Status codes	200, 400, 405, 415, 500, 503
Output	**txCredit** (required) The quantity of transactions still available on the module. Given as a positive integer.

Example of use #1 – XML response

Request

```
curl http://localhost:5081/stsvend/QueryTx.xml
```

Response

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<response>
  <txCredit>10000</priceExact>
</response>
```

Example of use #2 – TSV response

Request

```
curl http://localhost:5081/stsvend/QueryTx.tsv
```

Response

```
txCredit 10000
```

Example of use #3 – INI response (format not specified)

Request

```
curl http://localhost:5081/stsvend/QueryTx
```

Response

```
txCredit=10000
```

4.6 Calculate Watt-Hours (LEGACY)

Computes the quantity of electricity (hundred-Watt hours) that may be purchased for a given price.

URL	http://host/stsvend/CalcWh.*format*
Formats	tsv, ini *(default)*, xml
HTTP method(s)	POST
Input Parameters (Required)	**price** (required) The price that a customer will pay for the token, as a floating-point value in a standard currency unit (e.g. Rands). The decimal separator is a period (.). For example, “1.50” indicates one Rand and fifty cents.
	Tariff (required) The price for one unit of electricity (100 Watt-hours), as a floating-point value in the same currency unit and format as **price**. For example, “1.10” indicates one Rand and ten cents per hundred-Watt-hour.
	vendorFee (required) A per-vend fee that is charged by the vendor, to be subtracted from the price before applying the tariff. Given as a floating-point value in the same currency unit and format as **price**. For example, “0.75” indicates a fee of 75 cents.
Status codes	200, 400, 402, 405, 415, 500, 503
Output	**hwh** (required) The quantity of electrical units (hundred-Watt-hours) that may be purchased for the given price, tariff and fee. Given as a positive integer.
	priceExact (required) The exact price that should be paid for the quantity of electrical units indicated by **hwh**. **PriceExact** will always be less than or equal to **price**, and is given in the same currency unit and format as **price**. Since vending must occur in discrete units, each costing the amount given by **tariff**, the input **price** will seldom correspond exactly to a whole **hwh** quantity; that is, there will be a rounding error.
	priceCeil (optional) If **priceExact** is not equal to **price**, then **priceCeil** will indicate the price for (**hwh** + 1) electrical units; that is, the ceiling value with respect to the **tariff**.

Example of use #1 – XML response

Request

```
curl http://localhost:5081/stsvend/CalcWh.xml -d price=100 -d tariff=2 -d vendorFee=0
```

Response

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<response>
  <priceExact>100.00</priceExact>
  <hwh>500</hwh>
  <priceCeil>100.00</priceCeil>
</response>
```

Example of use #2 – TSV response

Request

```
curl http://localhost:5081/stsvend/CalcWh.tsv -d price=1000 -d tariff=1 -d
vendorFee=0.5
```

Response

```
priceExact	10000.00	hwh	99995	priceCeil	10000.00
```

Example of use #3 – INI response (format not specified)

Request

```
curl http://localhost:5081/stsvend/CalcWh -d price=45.5 -d tariff=1 -d vendorFee=0.75
```

Response

```
priceExact=45.45
hwh=447
priceCeil=45.55
```

Integration Guide

In this section we provide recommendations on how to use the services of the PrismVend API (section 4), and examples for some common use cases.

Each example develops an API concept; we urge you to read all examples, even those that may not seem relevant to your metering requirements.

STS tokens are not predictable: each token contains both time-variant and random data, and PrismVend detects and prevents duplicate tokens (as required by the STS standard).

Our examples include generated tokens, but you should not expect to see the same token values if you execute the example.

Recommended services

If you need to create Electricity, Water, Gas, Time, or Currency credit tokens for a given currency value then you should use the Vend Credit Token service (section 4.3). This service uses the PrismVend tariff tables (configured though the Web UI) to convert the currency value into appropriate metering units.

If you need to create Electricity credit tokens for a given Watt-hour value then you should use the Query Transaction Counter

Returns the number of transactions left in the module.

URL	`http://host/stsvend/QueryTx.`*`format`*
Formats	tsv, ini *(default)*, xml
HTTP method(s)	GET
Input Parameters (Required)	*None*
Status codes	200, 400, 405, 415, 500, 503
Output	**txCredit** (required) The quantity of transactions still available on the module. Given as a positive integer.

Example of use #1 – XML response

Request

```
curl http://localhost:5081/stsvend/QueryTx.xml
```

Response

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<response>
  <txCredit>10000</priceExact>
</response>
```

Example of use #2 – TSV response

Request

```
curl http://localhost:5081/stsvend/QueryTx.tsv
```

Response

```
txCredit 10000
```

Example of use #3 – INI response (format not specified)

Request

```
curl http://localhost:5081/stsvend/QueryTx
```

Response

```
txCredit=10000
```

service (section 4.5). There is no equivalent support for Water, Gas, or Time tokens.

If you need to create Meter-Specific Engineering (MSE) tokens you should use the Vend Meter-Specific Engineering Token service (section 4.4).

You should avoid using the following deprecated services:

- Calculate Watt-Hours (LEGACY)
- **Error! Reference source not found.**
- **Error! Reference source not found.**

4.7 Special warnings

The PrismVend Vending UI and the Web API behave slightly differently.

If you are getting different results from the UI and the Web API you should check the following:

- **The UI deducts transaction fees from the currency value before creating the token; the Web API does not. Check the transaction fees on the STS Administration page →Fees And Preferences (fields Prism Transaction Fee and Vendor Transaction Fee).**
- **The UI accepts a currency value in currency units (e.g. Rands, Dollars). The Vend Credit Token service accepts a currency value in $^{1}/_{100}$ currency units (i.e. "cents"). So if you entered $50.00 on the UI, you must use value=5000 when calling the service.**

4.8 Integration setup

The examples in this guide use the Compliance Test Specification (CTS) Test Keys. CTS is the standard used to test & certify STS vending systems.

To work with these examples *without modification* you will need a Security Module (SM) coded with the CTS MEK and KEK, and a Key Load File (KLF) containing the CTS Test Keys. You can get all of these from Prism.

If you prefer to use real meters and an SM coded with live keys (obtained in a KLF from a KMC) then you will need to replace the SGC and meter identification in the following examples with the values for your Supply Group and Meters.

We use the following Vending Keys (from the CTS KLF):

SGC	KRN	Type
123456	1	VUDK (Unique key per meter)

We use the following Meters (this table shows identification & configuration):

DRN	SGC	KRN	TI	EA	TCT	Type (Subclass)	IDRecord
00000000000	123456	1	01	07	02	Electricity (0)	60072700000000009000000207123456011
01316700887	123456	1	01	07	02	Water (1)	60072701316700887100000207123456011
0315000000002	123456	1	01	07	02	Electricity/ Currency (4)	00000315000000002600000207123456011

We use the following Tariff Tables (you must set up these tables in the PrismVend UI):

SGC	TI	Subclass (Type)	Rate
123456	01	0 (Electricity)	$1.24 / kWh
123456	01	1 (Water)	$1.50 / kL

The tariff table is expressed in local currency units. Credit vends must be denominated in the same units. We use the symbol '$' to indicate local currency, not a specific national currency.

Transaction fees will be deducted from currency values when vending through the PrismVend UI, but not when using the Web API's Vend Credit Token method (which is specified as using a value net of fees).

4.9 Electricity Credit Token, $50, Blind Vend

In this example we generate a credit token for an Electricity meter, with a credit value equivalent to 50 currency units (5000 "cents").

The meter is identified by a full 35-digit IDRecord (a "blind vend"). An IDRecord contains the meter's identification (the "MeterPAN") as well as the meter's configuration (SGC, KRN, TI, EA, and TCT). All this information is required by the vending system in order to create a token for the meter.

To generate this token the vending system will use the Vending Key for the meter's SGC and KRN. *In this example SGC=123456 and KRN=1.*

The vending system's Tariff Tables are used to convert the currency value ($50) into resource units understood by the meter (electricity meters only accept tokens with 100 Watt-hour units). The meter's SGC, TI, and type (subclass) are used to identify the relevant Tariff Table. *In this example the meter has SGC=123456, TI=01, and subclass=0, so the Tariff Table gives a rate of $1.24 / kWh, which translates to a 404 × 100Wh credit token.*

Note that the API requires the 'value' input to be in $^{1}/_{100}$ currency units ("cents"), so for a value of $50.00 we must use value=5000.

Example of use – XML response

Request

```
curl http://localhost:8080/stsvend/VendCredit.xml -d subclass=0 -d
meterId=600727000000000009000002O7123456011 -d value=5000
```

Response

```
<?xml version='1.0' encoding='utf-8'?>
<response><idRecord>600727000000000009000002O7123456011</idRecord>
<tariff>12.4</tariff>
<subclass>0</subclass>
<description>Sale - Credit:Electricity</description>
<vendTimeUnix>1458132240</vendTimeUnix>
<unitsActual>404</unitsActual>
<unitName>hWh</unitName>
<valueActual>5009.60</valueActual>
<tokenHex>28FE412A5E4F626F1</tokenHex>
<tokenDec>47261920893253068529</tokenDec>
</response>
```

The meanings of the response fields are described in the Vend Credit Token service (section 4.3). STS tokens are not predictable; you will get a different token each time to try this call.

4.10 Electricity Credit Token, $100, MSNO Vend

In this example we generate a credit token for an Electricity meter, with a credit value equivalent to 100 currency units.

The meter is identified by a DRN (a "meter serial number only (MSNO) vend"). The DRN is a shortened form of the MeterPAN (18 digits) that omits the IIN (the prefix "600727" or "0000") and the trailing PANCheckDigit. A DRN is thus 11 or 13 digits.

For an MSNO vend to work the vending system must know the meter's configuration (SGC, KRN, TI, EA, and TCT). The configuration may be known via a previous blind vend (e.g. the use case in section 0), or by entering the meter's configuration on the PrismVend UI.

As before the vending system must know the Vending Key (for SGC=123456 and KRN=1) and will use the Tariff Table (for SGC=123456, TI=01, and subclass=0) to convert the currency value into resource units for the meter.

Example of use – XML response

Request

```
curl http://localhost:8080/stsvend/VendCredit.xml -d subclass=0 -d meterId=00000000000
-d value=10000
```

Response

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<response><idRecord>60072700000000000900000207123456011</idRecord>
<tariff>12.4</tariff>
<subclass>0</subclass>
<description>Sale - Credit:Electricity</description>
<vendTimeUnix>1458132540</vendTimeUnix>
<unitsActual>807</unitsActual>
<unitName>hWh</unitName>
<valueActual>10006.80</valueActual>
<tokenHex>2A3440E4986998FDD</tokenHex>
<tokenDec>48658031982971293661</tokenDec>
</response>
```

The meanings of the response fields are described in the Vend Credit Token service (section 4.3). STS tokens are not predictable; you will get a different token each time to try this call.

4.11 Water Credit Token, $70, Blind Vend

In this example we generate a credit token for a Water meter, with a credit value equivalent to 70 currency units.

This example uses a different type of meter (Water) and thus a different Tariff Table (for SGC=123456, TI=01, and subclass=1) that translates from currency value in water resource units (100L). *The Tariff Table gives a rate of $1.50 / kL, which translates to a 467 × 100L credit token.*

The meter is identified by a full 35-digit IDRecord (a “blind vend”), and the vending system must know the Vending Key (for SGC=123456 and KRN=1).

Example of use – XML response

Request

```
curl http://localhost:8080/stsvend/VendCredit.xml -d subclass=1 -d
meterId=60072701316700887100000207123456011 -d value=7000
```

Response

```
<?xml version='1.0' encoding='utf-8'?>
<response><idRecord>60072701316700887100000207123456011</idRecord>
<tariff>15.0</tariff>
<subclass>1</subclass>
<description>Sale - Credit:Water</description>
<vendTimeUnix>1458132780</vendTimeUnix>
<unitsActual>467</unitsActual>
<unitName>hL</unitName>
<valueActual>7005.00</valueActual>
<tokenHex>3A9EC8DE22765BD9F</tokenHex>
<tokenDec>67584549710505295263</tokenDec>
</response>
```

The meanings of the response fields are described in the Vend Credit Token service (section 4.3). STS tokens are not predictable; you will get a different token each time to try this call.

4.12 Currency (Electricity) Credit Token, $120, Blind Vend

In this example we generate a 120 currency unit credit token for a Currency meter that dispenses Electricity.

An Electricity/Currency meter dispenses electricity but accepts credit tokens giving a Currency value rather than a resource unit value. To do this the meter has its own tariff table. The vending system's Tariff Tables are not used in this transaction; instead the currency value ($120) is encoded into the STS token.

The meter is identified by a full 35-digit IDRecord (a "blind vend"), and the vending system must know the Vending Key (for SGC=123456 and KRN=1).

Example of use – XML response

Request

```
curl -i http://localhost:8080/stsvend/VendCredit.xml -d subclass=4 -d
meterId=00000315000000002600000207123456011 -d value=12000
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 16 Mar 2016 13:59:53 GMT
Server: Prism WSHOST
set-cookie: nsssession=56e966d9.QlKNgocWYUPKveWE00TH_w; Path=/
content-length: 439
content-type: text/xml; charset=utf-8
cache-control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-check=0
expires: Sun, 01 Jul 2005 00:00:00 GMT
pragma: no-cache
vary: Accept-Encoding

<?xml version='1.0' encoding='utf-8'?>
<response><idRecord>00000315000000002600000207123456011</idRecord>
<tariff>1000</tariff>
<subclass>4</subclass>
<description>Sale - Credit:CurrencyElec</description>
<vendTimeUnix>1458136740</vendTimeUnix>
<unitsActual>12.00024</unitsActual>
<unitName>currency</unitName>
<valueActual>12000.24</valueActual>
<tokenHex>17F1A301544549985</tokenHex>
<tokenDec>27605429733819718021</tokenDec>
</response>
```

The meanings of the response fields are described in the Vend Credit Token service (section 4.3). STS tokens are not predictable; you will get a different token each time to try this call.

4.13 Currency (Electricity) Credit Token, Replay prevention

In this example we generate a 60 currency unit credit token for a Currency meter that dispenses Electricity, and we include the (optional) 'messageId' field in the request.

The ‘messageId’ is a unique transaction identifier that is used to detect and prevent replay of the transaction. If you have attempted a transaction but not received a response (for example there may be a timeout or a network interruption) then you cannot be sure whether the transaction was processed or not. In such a case you should resubmit the transaction with the same ‘messageId’

The ‘messageId’ is also useful to identify the origin of the transaction. Refer to Vend Credit Token (section 4.3) for recommended construction of this identifier.

Replay previous (using ‘messageId’) applies to all services, not just Vend Credit Token. We strongly recommend using it.

The meter is identified by a DRN. The vending system must know the meter’s configuration (SGC, KRN, TI, EA, and TCT; e.g. from use case 4.12) and the Vending Key required by that configuration (SGC, KRN). No Tariff Tables are required.

Example of use – XML response

Request

```
curl -i http://localhost:8080/stsvend/VendCredit.xml -d subclass=4 -d
meterId=0000031500000000260000207123456011 -d value=12000 -d messageId=POS.23.4-
emp0139-20130818T154023Z
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 16 Mar 2016 14:02:16 GMT
Server: Prism WSHOST
set-cookie: nsssession=56e96768.BUrRRLOomdLq_XowR0i2jg; Path=/
content-length: 439
content-type: text/xml; charset=utf-8
cache-control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-
check=0
expires: Sun, 01 Jul 2005 00:00:00 GMT
pragma: no-cache
vary: Accept-Encoding

<?xml version=’1.0’ encoding=’utf-8’?>
<response><idRecord>0000031500000000260000207123456011</idRecord>
<tariff>1000</tariff>
<subclass>4</subclass>
<description>Sale - Credit:CurrencyElec</description>
<vendTimeUnix>1458137040</vendTimeUnix>
<unitsActual>12.00024</unitsActual>
<unitName>currency</unitName>
<valueActual>12000.24</valueActual>
<tokenHex>0069D557A4050FE50</tokenHex>
<tokenDec>00476631119124561488</tokenDec>
</response>
```

The meanings of the response fields are described in the Vend Credit Token service (section 4.3). STS tokens are not predictable; you will get a different token each time to try this call.

4.14 Replayed transaction is rejected

In this example we repeat the API call from the previous example (section 4.13) using the same messageId, simulating a replay scenario.

Example of use – XML response

Request

```
curl -i http://localhost:8080/stsvend/VendCredit.xml -d subclass=4 -d
meterId=00000315000000002600000207123456011 -d value=12000 -d messageId=POS.23.4-
emp0139-20130818T154023Z
```

Response

```
HTTP/1.1 500 Internal Server Error

Date: Wed, 16 Mar 2016 14:03:10 GMT

Server: Prism WSHOST

set-cookie: nsssession=56e9679d.kJHcF2sBw_31asIHE4bXqg; Path=/

content-length: 353

content-type: text/html; charset=utf-8

cache-control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-
check=0

expires: Sun, 01 Jul 2005 00:00:00 GMT

pragma: no-cache

vary: Accept-Encoding


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<title>Internal Server Error</title>

</head>

<body>

<h1>Internal Server Error</h1>

<p>Vend credit operation failed. &#39;Message ID not unique: SQL query failed: UNIQUE
constraint failed: t_commits_cdu.commitRef&#39;</p>

</body>

</html>
```

4.15 Clear Credit Management Token, MSNO Vend

The Vend Meter-Specific Engineering Token service (section 4.4) is used to generate all management tokens.

In this example we create a “Clear Credit” management token. The token function (“Clear Credit”) is identified by subclass=1. For this function the ‘supp’ field is a bitmap indicating which credit registers to clear; we will use supp=65535 to clear all credit registers.

The meter is identified by a DRN (making this an “MSNO vend”). As with calls to Vend Credit Token we could have identified the meter by a full IDRecord (a “blind vend”). Since we identify the meter by DRN the vending system must know the meter’s configuration. In all cases the Vending Key corresponding to the meter’s configuration (SGC, KRN) is required to create the token.

Example of use – XML response

Request

```
curl http://localhost:8080/stsvend/VendMse -d subclass=1 -d meterId=00000000000 -d
supp=65535
```

Response

```
<?xml version='1.0' encoding='utf-8'?>
<response><idRecord>60072700000000000900000207123456011</idRecord>
<subclass>1</subclass>
<description>Tech - ClearCredit</description>
<vendTimeUnix>1458134460</vendTimeUnix>
<unitsActual>65535</unitsActual>
<unitName>regno</unitName>
<tokenHex>0CD3D1B9BF534F691</tokenHex>
<tokenDec>14789007108002346641</tokenDec>
</response>
```

The meanings of the response fields are described in the Vend Meter-Specific Engineering Token service (section 4.4). STS tokens are not predictable; you will get a different token each time to try this call.

5. Meter Management via the API

5.1 Background

In an STS system each meter has an identity (DRN/PAN), a configuration (SGC, KRN, TI), and a corresponding Meter Key.

The meter's identity is fixed for the lifetime of the meter; but the meter's configuration and the Meter Key can change over time, in particular by generating a Key Change Token Set and entering these tokens into the meter.

The fact that a meter's configuration can change, and that the change is split into a token generation step and a token loading step, means that there are always at least 3 possible things that a "meter configuration" could refer to:

1. The actual SGC, KRN, TI and Key of the meter. We call this the "actual configuration".
2. The SGC, KRN, and TI recorded for the meter in the Vending System. We call this the "captured configuration".
3. The SGC, KRN, and TI that the meter *should be configured on* according to the system management policies. We call this the "intended configuration".

The Vending System also distinguishes between "Registered" and "Unregistered" (also called "Seen") meters:

- If a meter is "Registered" then the Vending System asserts that its view of the meter configuration (SGC, KRN, TI) is accurate. Attempts to "blind vend" to the meter using a different SGC/KRN/TI will fail.
- If a meter is "Unregistered" (also called "Seen") then the Vending System will trust the SGC/KRN/TI provided in a "blind vend", issue tokens using that configuration, and store it to the database for later use.

Understanding that there are different views of "meter configuration", and how blind vending to an "Unregistered" meter can change the meter configuration stored in the database, is key to understanding how to manage meter configuration in the Vending System and in the meter.

5.2 Get Meter Information (GET/stsvend/Meter/*drnOrPan.format*)

Retrieves meter information from the database.

Use cases:

- Get meter configuration (and other information) for display outside the PrismVend WebUI.
- Get information for subsequent update via "Set Meter Information"

Prerequisites:

- Meter must exist in database

Description:

Retrieves meter configuration and other information from the PrismVend database. The same information can be seen in the PrismVend WebUI. Information includes the meter configuration, descriptive names, tariff, and applicable channel fees.

Required inputs:

- drnOrPan: Meter DRN or PAN

Optional inputs:

- format: tsv|ini|xml, default ini

Outputs:

- HTTP status code 415 "Unsupported Media Type": bad “format” input
- HTTP status code 400 "Bad Request": bad input (e.g. invalid drnOrPan)
- HTTP status code 404 "Not Found": meter could not be found in database
- HTTP status code 500 “Internal Server Error”: unhandled exception
- HTTP status code 200 plus TSV/INI/XML body with the following fields: docId, idRecord , meterPan, drn, sgc, krn, ti, ea, tct, doe, isSeen, isRegistered, resType, name, organisation, channelFee, tariffRate, balanceCf, queuedKct.
    - A meter `isRegistered` if its configuration has been entered via the WebUI or using “Set Meter Information”. A meter will not be registered if its configuration has only been recorded via a “blind vend”. A meter configuration must be registered before a different intended configuration can be recorded.
    - balanceCf is not documented in PR-D2-1112
    - queuedKct is not documented in PR-D2-1112: it is empty if no queued KCT (no intended configuration), otherwise a space-separated key-value list of SGC, KRN, and TI.

Testing:

```
Moved to HLSM/tests/cdu-http-kct.tcl


curl -i http://localhost:8080/stsvend/

HTTP/1.1 200 OK

<!DOCTYPE html><html><head><title>TSM Status</title></head><body><h1>stswebApi Service
Status</h1><table class="status"><tr><td>Network
address</td><td>127.0.0.1</td></tr><tr><td>Web UI port</td><td>80</td></tr><tr><td>Web
service port</td><td>8080</td></tr><tr><td>TSM family</td><td>Prism-TSM250-
1</td></tr><tr><td>TSM UID</td><td>FFFFFF10000001FF</td></tr><tr><td>TSM
firmware</td><td>STS65 v1.0.0.0</td></tr><tr><td>Access control mode</td><td
class=badvalue>AC:NONE</td></tr><tr><td>Performance
rating</td><td>TPS:9999</td></tr><tr><td>Up since</td><td>2023-05-03
17:29:28</td></tr></table></body></html>

This is what is implemented in `proc stswebApi::/` and it’s not limited to APP_DEBUG.
So this is the correct response, but is it the desired response?


curl -i http://localhost:8080/stsvend/dump

HTTP/1.1 200 OK

-acceptTime=1683128901391112

-acmCmd=::acm

…

OK; this is only available if APP_DEBUG=1 at startup


curl -i http://localhost:8080/stsvend/Meter/00000000000.ini

HTTP/1.1 200 OK

set-cookie: nsshttp=64528343.6pK4eRpf1WYYMWr5nBfx7g; Path=/; HttpOnly

cache-control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-
check=0
```

```
expires: Sun, 01 Jul 2005 00:00:00 GMT

pragma: no-cache

content-type: text/plain; charset=utf-8

X-Frame-Options: sameorigin

X-XSS-Protection: 1; mode=block

X-Content-Type-Options: nosniff

Content-Security-Policy: default-src 'self' 'unsafe-inline' 'unsafe-eval'; connect-src
*; img-src 'self' data:

content-length: 266

vary: Accept-Encoding


meterPan=600727000000000009

sgc=123456

krn=1

ti=1

ea=7

tct=2

idRecord=60072700000000000900000207123456011

doe=0000

docId=600727000000000009;206

drn=00000000000

isSeen=1

isRegistered=1

resType=-1

name=

organisation=

balanceCf=0.04

channelFee=R0.25

tariffRate=NOT SET

200 + body is OK

Cache headers (cache-control, expires, pragma) OK

Security headers (X-Frame-Options, X-XSS-Protection, X-Content-Type-Options, Content-
Security-Policy) OK

Set-cookie: why do we have cookies on the Web API?

resType=-1 shows on the UI as Electricity(0)


# Meter not in database

curl -i http://localhost:8080/stsvend/Meter/99000999997.tsv

HTTP/1.1 404 Not Found

…<p>Meter not found in database</p>…

OK
```

5.3 Set Meter Information (POST/stsvend/Meter/*drnOrPan.format*)

Creates or updates a meter record in the database.

Use cases:

- Correct the captured configuration (in the database) to match the actual configuration (of the meter).
- Queue a Key Change Token for delivery to a meter along with the next Credit or MSE token, by setting an intended configuration (in the database) that is different to the captured configuration (in the database).
- Integrate with a front-end UI (other than PrismVend Web UI).
- Automate meter changes via scripts or as part of a batch operation.
- Change meter configuration in database to match the actual meter

Prerequisites:

- For an existing meter: use “Get Meter Information” first to get the current `docId`.

Description:

Creates a new meter record in the database (capturing the meter’s configuration), or updates the configuration (and/or other information) of an existing meter record in the database.

Note:

- Changing the meter to any well-formed SGC, KRN, and TI will succeed, even if the Vending Key is not available to the Vending System, or the SGC/KRN/TI violate policies. A subsequent attempt to issue a Key Change Token will apply the relevant policies (if any), and will fail if the Vending Key is not available.

Required inputs:

- drnOrPan: Meter DRN or PAN
- docId: an optimistic lock value which must be `new` for a new meter record, or the `docId` value returned by “Get Meter Information” for an existing meter record.

Optional inputs:

- format: tsv|ini|xml; default ini
- resType, sgc, krn, ti: Required for a new meter, optional for an existing meter.
- messageId, ea, tct, name, organisation, balanceCf: Always optional.
- newSgcKrnTi: capture|queuekct. Default is `capture` which means the SGC/KRN/TI is stored to the database as a correction to the existing record (i.e. to make the captured configuration match the actual meter configuration). `queuekct` will only work on a meter that is already registered, and sets the intended configuration, which (if different to the captured configuration) will cause a Key Change Token to be issued as part of the next Credit or MSE vend (changing the meter from the captured configuration to the intended configuration).
- *All other fields names (including other fields returned by “Get Meter Information” like idRecord, isSeen, isRegistered, channelFee, tariffRate, etc.) are ignored.*

Outputs:

- HTTP status code 415 "Unsupported Media Type": bad “format” input
- HTTP status code 400 "Bad Request": a required input field is missing, or any input field has an invalid value
- HTTP status code 404 "Not Found": meter could not be found in database
- HTTP status code 409 "Meter Conflict": wrong docId value (maybe because the document has changed since you retrieved the docId value)
- HTTP status code 500 “Internal Server Error”: unhandled exception
- HTTP status code 200 plus TSV/INI/XML body as for Get Meter Information.

Testing:

```
Moved to HLSM/tests/cdu-http-kct.tcl


# Set REV to the rev in docId

curl -s -i http://localhost:8080/stsvend/Meter/00000000000.ini | findstr docId

set REV=rev


# Basic success case

curl -i http://localhost:8080/stsvend/Meter/00000000000.ini -X POST -d
"docId=600727000000000009;%REV%" -d "name=STSA Meter" -d "organisation=STSA"

OK


# Investigate default “capture/set” and how it shows up in the UI

set /a REV=REV+1 && echo.

curl -i http://localhost:8080/stsvend/Meter/00000000000.ini -X POST -d
"docId=600727000000000009;%REV%" -d "sgc=888888" -d "krn=1" -d "ti=44"

In UI: History = Meter Registered (Tech). Meter details: TI '44', KRN '1' → so
default is “capture” → OK


# Investigate explicit “capture/set” and how it shows up in the UI

set /a REV=REV+1 && echo.

curl -i http://localhost:8080/stsvend/Meter/00000000000.ini -X POST -d
"docId=600727000000000009;%REV%" -d "sgc=888888" -d "krn=1" -d "ti=44" -d
"newSgcKrnTi=capture"

History = Meter Registered (Tech). Meter details: TI '44', KRN '1' → OK


# Investigate “queuekct” and how it shows up in the UI

set PARAMS=-d "sgc=888888" -d "krn=1" -d "ti=99" -d "newSgcKrnTi=queuekct"

set /a REV=REV+1 && echo.

curl -i http://localhost:8080/stsvend/Meter/00000000000.ini -X POST %PARAMS% -d
"docId=600727000000000009;%REV%"

History = Meter Registered. Meter details: TI '99', KRN '1' → OK


# How does queuekct show up in the API?

curl -i http://localhost:8080/stsvend/Meter/00000000000.tsv

HTTP/1.1 200 OK

meterPan        600727000000000009      sgc     888888  krn     1       ti      99
ea      7       tct     2       idRecord 60072700000000009000002078888889991     …

OK
```

```
Note that the API shows the intended configuration (newIdRecord / queued KCT).
Checking the UI reveals that it too shows the intended configuration. The database
content can be revealed by `::stsAdminZone::cdu debugGetMeterRecord 00000000000`,
which has `… idRecord 600727000000000090000207888888441 newIdRecord
600727000000000090000207888888991 …`.

Note that the captured configuration (idRecord) cannot be seen in the UI or API when
an intended configuration (newIdRecord) is present. That's probably not right!
```

5.4 Update Meter Key (POST/stsvend/UpdateMeterKey.*format*)

Issues a Key Change Token Set to change the meter's configuration to match the system.

Use cases:

- Meter configuration does not match system; issue KCT to change meter to match the system.
- System has queued KCT (non-empty newIdRecord), issue KCT to change meter to new configuration.
- System has received new KRN for SGC or new KEN for VK, issue KCT to change meter KRN or KEN (CTS auto-KCT scenario).

Prerequisites:

- Meter must exist in database

Description:

This function implements the SANS 1524-6-10 "Update Meter Key" process: it issues a KCT from the input SGC/KRN/TI values, to the "intended" meter configuration stored in the database (i.e. queuekct), or to the "current" configuration if there is no "intended" configuration.

Notes:

- The operation may fail if the VendingKe y for the destination SGC & KRN is not available.
- Failing to enter a KCT (issued by this endpoint) into a Meter will cause the meter's actual configuration to different from the system's current configuration.
- The actual meter configuration (fromSgc, fromKrn, fromTi) is usually taken from the receipt of a token that has been accepted by the meter. If the system's current meter record is accurate, this can be retrieved using `GET /Meter`.
- By default a KCT is only issued if the meter needs an updated key, so you can use UpdateMeterKey to determine if an update is required *so long as you are prepared to receive a KCT and enter it into the meter*. To determine whether an update is available without issuing a KCT you can use `GET /Meter`, then an update if required if (1) the actual meter configuration differs from the idRecord or (2) there is a non-empty queuedKct.

Required inputs:

- meterId: Meter DRN or PAN (IDRecord not accepted)
- fromSgc, fromKrn, fromTi: configuration of the physical meter

Optional inputs:

- format: tsv|ini|xml; default ini
- messageId

- skipIfSameMeterKey (default 1): By default a KCT will not be generated if the given meter configuration already matches the system; set to 0 to force a KCT to be issued in this circumstance.

Outputs:

- HTTP status code 200 plus TSV/INI/XML body with the following fields:
  - description (string): Human-readable description of the operation
  - vendTimeUnix (int): Date/time of issue as a Unixtime (seconds since 1970-01-01)
  - fromIdRecord (IDRECORD): Source/from configuration, which will match the supplied meterId, fromSgc, fromKrn, and fromTi.
  - toIdRecord (IDRECORD): Destination/to configuration, which will match the system.
  - isRegistered
  - numTokens (int): Number of tokens
  - tokenDec_1, tokenDec_2, … tokenDec_{numTokens} (20 digits): 2 or more tokens (as separate fields) that make up the Key Change Token Set.
  - description_1, description_2, … description_{numTokens} (string): For each `tokenDec_{N}` there will be a matching `description_{N}` that describes the token (can be used in receipt printing).
- HTTP status code 204 “No Content”: meter does not require an updated key (source and destination configurations are the same) so no KCT was issued
- HTTP status code 400 "Bad Request": a required input field is missing, or any input field has an invalid value
- HTTP status code 402 " Payment Required": insufficient transactions in SM to complete the operation
- HTTP status code 415 "Unsupported Media Type": bad “format” input
- HTTP status code 422 " Unprocessable Entity ": well-formed request but could not be processed, most likely because the meter could not be found in database
- HTTP status code 500 “Internal Server Error”: unhandled exception

Testing:

```
Moved to HLSM/tests/cdu-http-kct.tcl


# Basic success

curl -i http://localhost:8080/stsvend/Meter/00000000000

# Change SGC,KRN,TI if needed so that the from* below are different to the database
configuration

curl -i http://localhost:8080/stsvend/UpdateMeterKey.ini -X POST -d
"meterId=00000000000" -d "fromSgc=888888" -d "fromKrn=1" -d "fromTi=1"

HTTP/1.1 200 OK
```

5.5 Engineering Key Change (POST/stsvend/EngineeringKeyChange)

Change the meter's intended configuration on the system, then issue a KCT to change the meter's actual configuration to match the system.

Use cases:

- Meter not in database; add meter to database with "to" configuration, then issue a KCT for the given "from" and "to" values.
- Meter in database (current configuration – and intended configuration if present - may or may not be correct); issues a KCT for the given "from" and "to" values, sets the current configuration in the database to the "to" values, and clears any queued KCT.

Prerequisites:

- None

Description:

This function implements the SANS 1524-6-10 "Engineering Key Change" process, but with relaxed limits on the destination configuration. First the input toSgc/toKrn/toTi values will be used to set the intended configuration of the meter in the database (to the extent permitted by policies & configurations); then a KCT is issued from the input fromSgc/fromKrn/fromTi values to the intended configuration. The current and/or intended meter configuration that existed in the database before this function was called is ignored, except to provide EA and TCT values if those were not supplied as function parameters.

Once this KCT has been loaded into the meter, the meter's actual configuration should be in sync with the Vending System.

This function is the equivalent of using `POST /Meter` to change the current configuration, then `POST /UpdateMeterKey` to issue a KCT from the meter's actual configuration to the (new) configuration on the system.

Notes:

- Use this function when a KCT must be sent to the meter and the configuration in the database must be changed.
- The actual meter configuration (fromSgc, fromKrn, fromTi) is usually taken from the receipt of a token that has been accepted by the meter. If the system's current meter record is accurate, this can be retrieved using `GET /Meter`.
- Optional parameters EA and TCT are used in preference to those stored in the database, will update the database values for EA & TCT for this meter, and are required if the meter is not in the database.
- Policies/rules/configurations may override the toSgc and/or toKrn and/or toTi. For example: an "autoUpdateKrn" rule would always select the most-recently activated KRN for the SGC, irrespective of toKrn; the effective TI may be dictated by a Tariff. The technical operator is prevented from breaking these policies.
- The operation may fail if the VendingKey for the destination SGC & KRN is not available.
- Failing to enter a KCT (issued by this endpoint) into a Meter will cause the meter's actual configuration to different from the system's current configuration.

- The function is similar to the PrismVend UI's "Key Change" tab, except that you can also specify the meter's "from" configuration.

Required inputs:

- meterId: Meter DRN or PAN (IDRecord not accepted)
- fromSgc, fromKrn, fromTi: actual configuration of the physical meter
- toSgc, toKrn, toTi: target configuration of the KCT, also stored to the database as the meter's current configuration.

Optional inputs:

- format: tsv|ini|xml
- messageId
- toEa, toTct: actual EncryptionAlgorithm and TokenCarrierType of the meter to use when issuing the KCT, and to be stored to the database as part of the meter's current configuration. Required if the meter is not in the database; other defaults to the meter's current values in the database.

Outputs:

- HTTP status code 200 plus TSV/INI/XML body with fields as for /UpdateMeterKey.
- HTTP status code 400 "Bad Request": a required input field is missing, or any input field has an invalid value
- HTTP status code 402 " Payment Required": insufficient transactions in SM to complete the operation
- HTTP status code 415 "Unsupported Media Type": bad "format" input
- HTTP status code 422 " Unprocessable Entity ": well-formed request but could not be processed, most likely because the meter could not be found in database and toEa/toTct were not supplied
- HTTP status code 500 "Internal Server Error": unhandled exception

Testing:

```
Moved to HLSM/tests/cdu-http-kct.tcl
```

6. Amendment History

Version	Description	Person	Date
3.9.0	Duplicate of PR-D2-0835 (StsClosedVending API).	TD	2013/03/06
3.29.0	Document adapted from PR-D2-0835 “Prism STS Vending Web Service” (rev 3.9.0). Updated to indicate legacy services, and to document the “Vend Credit” service.	TD	2013/09/05
3.32.0	Added “Vend Meter-Specific Engineering Token” service.	TD	2013/10/24
4.17	Added table of management functions to Vend Meter-Specific Engineering Token. Added the Integration Guide and examples.	TD, CA	2016/03/14
4.17.1	Fixed description of ‘idRecord’ output field in VendCredit.	TD	2016/04/29
4.23.8	Updated docs for Vend Credit Token, in particular the response fields unitName and valueActual.	TD	2016/07/22
4.27.4	Removed references to outdated readme	CA	2015/09/30
4.27.12	Added new subclass value to the VendCredit API function. If -1 is specified the CDU will perform a database look up to determine the resource type of the registered meter	CA	2017/01/29
4.29	Added a new call \QueryTx, which queries the transaction counter	CM	2017/02/04
4.35	Changed the description of ‘Meter Key Change’ under ‘Supported Subclasses’ to point to the User Interface	CM	2017/03/23
4.36	Added terms, definitions and abbreviations section	CA	2017/04/28
4.37	Removed all legacy API commands that are either unsupported or have replacement commands.	CA	2017/12/11
4.38	Fixed all curl examples so that they can be copied, pasted and executed in a console.	CA	2018/08/23
4.40	Added /Meter endpoint which allows the user to query a specific meters configuration	CA	2021/11/25
5.20	Added support for KCT over the API. STS6 only	CA	2023/05/21