Tel: +27 11 343 2000 | **Fax**: +27 11 442 5908 | **Email**: info@prism.co.za
Address: President Place, Johannesburg, South Africa

PrismToken Thrift API

2024-07-01

Document number:	PR-D2-1009 Rev 1.0.11
Release date:	2024-07-01
Prepared by:	TrevorD
Copyright:	© 2024 Prism Payment Technologies
Synopsis:	Specifies the Application Programming Interface (API) for PrismToken.

Company Confidential

Disclaimer

Contents

1 Introduction

PrismToken provides a network service and remotely callable Application Programming Interface (API) to issue STS tokens.

The network service is accessible over TCP/IPv4 using Thrift. Transport Layer Security (TLS) is required.

The API is specified using the Thrift Interface Definition Language (IDL), from which client software can be automatically generated.

1.1 PrismToken overview

Refer to section “PrismToken” in “Introduction to STS and PrismToken” (PR-D2-1060 Rev 1.1).

1.2 Thrift overview

From Wikipedia: "Thrift is an interface definition language (IDL) and binary communication protocol that is used to define and create services for numerous languages. It is used as a remote procedure call (RPC) framework and was developed at Facebook for "scalable cross-language services development". It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, Node.js, OCaml, Perl, PHP, Python, Ruby and Smalltalk. Although developed at Facebook, it is now an open source project in the Apache Software Foundation"

Advantages of using Thrift include:

- The IDL precisely specifies the API, and facilitates automatic generation of client libraries (for supported languages).
- Thrift is portable across platforms and operating systems, and supports a wide range of programming languages.
- Apache Thrift is open source and actively developed. There are no licensing costs.
- The Thrift specification covers protocol (wire format), transport (wire endpoints), and a Remote Procedure Call framework. This standardises the full communication stack.
- Thrift scales well.

1.3 References

[IEC 62055-41]	"Standard transfer specification (STS) – Application layer protocol for one-way token carrier systems", Ed 1 Sep 2007
[IEC 62055-51]	“Standard transfer specification (STS) – Physical layer protocol for one-way numeric and magnetic card token carriers”, Ed 1 Aug 2007
[SANS 1524-6-10]	“Electricity payment systems – Part 6-10: Interface standards – Online vending server – Vending clients”, Ed 1 Nov 2010
[STS 202-1]	"Addendum to IEC62055-41 - Currency Token", June 2014

[STS 402-1]	"Code of Practice - Management of Token ID Rollover", Ed 1 Mar 2014
[STS 600-4-2]	"Standard Transfer Specification - Companion Specification - Key Management System", Ed 1.2 Mar 2015

1.3.1 Obtaining reference documents

Reference documents (other than Prism documents) are Copyright by third parties and cannot be supplied by Prism.

- IEC documents have a reference code prefix [IEC], and may be obtained from the International Electrotechnical Commission (IEC) (http://www.iec.ch/), or your national affiliate (for example SABS in South Africa).
- South African National Standard (SANS) documents have a reference code prefix [SANS], and may be obtained from the South African Bureau of Standards (SABS) (https://www.sabs.co.za/).
- STS Association (STSA) documents have a reference code prefix [STS]. Members of the Association may obtain these documents from the STSA website (http://www.sts.org.za/).
- Prism documents have a reference code prefix [PR-], and may be obtained from Prism.

2 PrismToken test configuration

2.1 Preferences

The following PrismToken preferences may assist when developing a client for the Thrift API:

hlsm.allowUnapprovedKmcPubkeys	Allows individual PUBKEYs to be loaded (as opposed to only an STSA Config file).
hlsm.allowKeyChangeToFutureBDT	Allow generating a Key Change Token Set where the destination key's Base Date is in the future. Some of the STS531 tests cannot pass unless you enable this setting.

2.2 Obtaining deterministic outputs

Token Issue is not normally deterministic: PrismToken protects against the issue of duplicate tokens, and most tokens include random data that is set by the Security Module.

It is possible to set up a test environment to obtain deterministic outputs, which are useful in development and automated testing:

- You must obtain a Key Load File from Prism's 'PRISM-TEST-KMC' key management system. This system can supply you with STS531 (CTS) keys with special behavior: token generation does not alter the transaction counter, and the RND field of any token is set to 5;
- You must use the EXTERNAL_CLOCK flag when calling the API; and
- To allow duplicate tokens to be issued you will need to call ctsResetTidList to clear the TokenCancallation list for one or more meters.

3 Thrift Server

3.1 Transport & Protocol

The PrismToken Thrift server uses **TFramedTransport** over **TTLSSocket** (Transport Layer Security over TCP/IPv4). The default TCP port is 9443.

The server accepts and sends messages encoded in **TBinaryProtocol** only.

Make sure you are using TFramedTransport.

Make sure you are using TBinaryProtocol.

3.1.1 Transport Layer Security (TLS)

The PrismToken Thrift server requires Transport Layer Security (TLS), and mandates the use of protocol version 1.2 with secure cipher suites.

Your client software must use TLS 1.2.

The original Thrift 0.9.3 distribution (Oct 2015) may not support TLS 1.2 for your development language. You may need to use a more recent version from the Thrift repository, or to modify the Thrift sources (choose a Framework/library version that supports TLS 1.2, and set the appropriate constants/flags in TTLSSocket).

For C# or .NET development on Windows:

- **Develop and run on Windows 8.1 or higher, or Windows Server 2012 R2 or higher, with the latest updates.**
- **Use .NET Framework 4.7 or higher, with the latest patch of the targeting pack.**
- **Use Visual Studio 2017 or higher, with the latest updates.**

Older versions of the OS, Framework, or Visual Studio need workarounds to use TLS 1.2 as explained by Microsoft in Transport Layer Security (TLS) best practices with the .NET Framework. Building or running an application with these older versions will likely produce an AuthenticationException or SSPI error; the Program.cs in our C# DevPack has detailed notes on this problem.

3.2 Access control and authentication

Some of the methods exposed by the service are protected by access control. These methods require an 'accessToken' parameter.

An 'accessToken' can be obtained from any of the 'signIn' methods (such as signInWithPassword). The `accessToken` is specific to a PrismToken instance, not to a network connection. If you have multiple connections to a PrismToken instance they can all use the same `accessToken`. If you are load balancing across multiple PrismToken instances then you must use the correct `accessToken` for the instance!

The 'accessToken' is short-lived and is sensitive to disclosure – you should not store the accessToken is communicate it over an insecure network connection. Because the token is short-lived you will need to obtain a fresh token from time to time:

- You can control the accessToken's longevity using the `ptoken.sessionTimeoutSec` preference (default 300 seconds, maximum 24 hours).
- Our standard recommendation for managing `accessToken` expiry is that after any `EAuthentication.Token` exception you obtain a fresh accessToken (from signInWithPassword) and re-try the call.
- A variant on the above is that you track the age of the accessToken, and obtain a fresh accessToken before it expires (`ptoken.sessionTimeoutSec` seconds). Note that this cannot handle cases where the accessToken is invalidated for reasons other than age (e.g. if the PrismToken instance reboots).
- Another variant is to test the validity of the `accessToken` by calling a side-effect-free API method like `parseIdRecord()`. Again there are cases that this cannot handle, such as the race condition where the token expires between the validity test call and the subsequent call.
- We strongly recommend against obtaining a new `accessToken` for each request: passwords are protected by a stretched hash that is slow to compute, and will reduce your overall throughput substantially.

3.3 Creating a Thrift Client

In order to use the client library to communicate with the Thrift server the following steps need to be followed in your programming language:

1. Create a TTLSSocket object with IP address and port (default port is 9443) of the Thrift server to connect to.
2. Create the TFramedTransport object using the TTLSSocket object
3. Create the TBinaryProtocol object using the TFramedTransport object
4. Create the client object using the TBinaryProtocol object

4 Thrift types

The Exception and Struct types used by PrismToken are defined in the Thrift IDL file which is supplied with this document. This section details the purpose of each field.

4.1 Constants

4.1.1 ApiVersion

API version number, use in SessionOptions at sign-in. Server 1.MAX works with client 1.MIN; Client 1.X requires server 1.X or higher.

4.1.2 PrintableAsciiRe

Printable ASCII is a character alphabet defined in STS600-4-2. This const gives the POSIX extended regular expression that matches a printable ASCII string (zero or more characters in the range x'20 to x'7E (ASCII)).

TTY ASCII is a character alphabet defined for this API; it is printable ASCII plus the LF (line feed, ASCII x'10) control character.

4.1.3 IdentMatchRe

An IDENT is a type defined in STS600-4-2; this const gives the POSIX extended regular expression that matches an IDENT. An IDENT comprises one alphanumeric character (A-Z, a-z, or 0-9) followed by 0 to 39 characters that are either alphanumeric or underscore (ASCII x'5F), hyphen (ASCII x'2D), period (ASCII x'2E) or comma (ASCII x'2C). An IDENT is a printable ASCII string with an overall length of 1 to 40 characters.

4.1.4 ApiErrors

Map of error identifiers (key) to English error descriptions (value). Keys have range IDENT; values have range TTY ASCII (printable ASCII plus LF).

4.2 Enums

4.2.1 TokenIssueFlags

Bitmap of flag values supported in methods issueCreditToken and issueMseToken:

- EXTERNAL_CLOCK: The method's tokenTime input is a Unixtime (UTC) to be used as the effective clock. Mutually exclusive with TID_ADJUST_BDT.
- TID_ADJUST_BDT: The method's tokenTime input is a minute offset from BDT=1993 (1993-01-01 00:00 UTC) to use as the TokenID. The value is adjusted to be relative to the VendingKey's BDT, but _is not_ adjusted for TokenCancellation or to skip the SpecialReservedTokenIdentifier. Cannot be used with either EXTERNAL_CLOCK or SPECIAL_RESERVED.
- SPECIAL_RESERVED: Adjust the TokenID to be the SpecialReservedTokenIdentifier (time 00:01) at the start of the day indicated by the system clock (or EXTERNAL_CLOCK). Mutually exclusive with TID_ADJUST_BDT.

4.3 ApiException (exception)

Exception class used by all methods in this API. The cause or nature of the exception is indicated by the field 'eCode' which allows the client to respond to specific error conditions and to localise the error message.

ID	Presence	Type	Name	Description
1	**required**	**string**	**eCode**	Identifies the error and its categorisation. Range: IDENT (should be a key from ApiErrors, but clients cannot rely on this as they may be compiled against an older API than the server). A list of codes is given in section 4.3.1.
2	**required**	**string**	**eMsgEn**	A human-readable message explaining the error, in English. Range: TTY ASCII (printable ASCII plus LF (line break).

4.3.1 ApiException error codes

See ApiErrors.

4.4 SessionOptions (struct)

Session options that are common across sign-in methods.

ID	Presence	Type	Name	Description
1	**required**	**string**	**version**	Client must set this field to ApiVersion (format is VerMajor.VerMinor).
10	**optional**	**string**	**culture**	Culture is an IETF language tag per RFC 4646, but we only accept the forms 'LL' or 'LL-CC' where LL is an ISO 639-1 two-letter language code and CC is an ISO 3166-1 alpha-2 (two-letter) country code, e.g. 'en' or 'en-ZA'.

4.5 SignInResult (struct)

Result type for all sign-in methods.

ID	Presence	Type	Name	Description
1	**required**	**string**	**accessToken**	A bearer token that proves the client's authority. Client must treat this value as opaque and keep it secret. Range: printable ASCII. See Access control and authentication (section 3.2).

4.6 Alert (struct)

An identifiable (by a code) error/warning/alert condition that has a description and can be localised.

ID	Presence	Type	Name	Description
1	**required**	**string**	**eCode**	Identifies the error and its categorisation. Range: IDENT (should be a key from ApiErrors, but clients cannot rely on this as they may be compiled against an older API than the server). A list of codes is given in section 4.3.1.

ID	Presence	Type	Name	Description
2	required	string	eMsgEn	A human-readable message explaining the error, in English. Range: TTY ASCII (printable ASCII plus LF (line break).

4.7 NodeStatus (struct)

Status of a node in a tree network of PrismToken instances (method getStatus() returns a list of these).

ID	Presence	Type	Name	Description
1	required	map<string, string>	info	Associative array of status information. Keys have range IDENT, values are printable ASCII
2	required	list<Alert>	alerts	Zero or more alert messages

A selection of pertinent keys returned in the info map

Type	Name	Description
string	HardwareId	SM hardware identifier per STS600-4-2
string (8 digits)	ModuleId	SM serial number
string	FirmwareId	SM firmware identifier (including firmware revision) per STS600-4-2
string	ApiType : "vending" or "manufacturing"	The STS API type implemented by the SM
non-negative decimal integer	TxCounter	Number of transactions remaining before SM's transaction license expires, unlimited=2,000,000,000.
ISO 8601 point-in-time string	smqd.ex.RtcTimeUtc	Current date & time according to the SM's Real-time Clock (YYYYMMDD 'T' HHMMSS 'Z')
ISO 8601 point-in-time string	smcq.ex.InhibitVendDateUtc	Date & time at which the SM's transaction license expires
string	smud.Uid	SM hardware unique identifier. This is distinct from the ModuleId. You should generally use the ModuleId to identify the SM.
string	smqi.IdSm	SM "PKID" – the SM's fully-qualified identity (including serial number and public key fingerprint) per STS600-4-2.

4.8 MeterConfigAmendment (struct)

Carries the new meter configuration (to be established by the PrismToken instance issuing a KCT) when a change has been captured on the Vending System.

ID	Presence	Type	Name	Description
1	required	i32	toSgc	New SupplyGroupCode, range 0-999999
2	required	i16	toKrn	New KeyRevisionNumber, range 1-9
3	required	i16	toTi	New TariffIndex, range 0-99

4.9 MeterConfigIn (struct)

Identifies the meter configurtion in token-issuing calls.

In a typical STS environment the DRN is the 11- or 13-digit serial number of the meter, and the meter configuration fields (SGC, KRN, TI, EA, TCT) are captured into the Vending System's database when the meter is deployed. In some environments the meter identity and configuration may come from a magstripe card or may be stored in a database as an IDRECORD; in these cases method parseIdRecord may help to get a MeterConfigIn from the data).

ID	Presence	Type	Name	Description
1	required	string	drn	DecoderReferenceNumber; 11- or 13-digit DRN, or 18-digit PAN, with valid Luhn(s).
2	required	i16	ea	EncryptionAlgorithm, range 0-99; only 7,11 currently supported
3	required	i16	tct	TokenCarrierType, range 0-99
10	required	i32	Sgc	SupplyGroupCode, range 0-999999
11	required	i16	Krn	KeyRevisionNumber, range 1-9
12	required	i16	Ti	TariffIndex, range 0-99
20	optional	MeterConfigAmendment	newConfig	Struct containing the new intended configuration. See also 7.1 Automatic Key Change Tokens.
21	optional	bool	allowKrnUpdate	Issue KCT when newer KRN is available for the SGC.Default 'true'. See also 7.1 Automatic Key Change Tokens.
30	required	i16	ken	KeyExpiryNumber, range 0-255

ID	Presence	Type	Name	Description
31	optional	string	Doe	Date of expiry of the meter card. Range 4 digits giving 'YYMM'. Default '0000' which implies unused.
32	optional	bool	allow3Kct	(Only for EA=07) Use 3-token set when issuing key change tokens. Default 'false'
33	optional	bool	allowKenUpdate	Issue KCT when meter KEN differs from VK KEN. Default 'true'. See also 7.1 Automatic Key Change Tokens.

4.10 MeterConfigAdvice (struct)

Carries the new meter configuration to be stored by the Vending System as a consequence of a Key Change Token Set being issued to the meter.

ID	Presence	Type	Name	Description
1	required	i32	toSgc	New SupplyGroupCode, range 0-999999
2	required	i16	toKrn	New KeyRevisionNumber, range 1-9
3	required	i16	toTi	New TariffIndex, range 0-99
4	required	i16	token	New KeyExpiryNumber, range 0-255
10	required	string	idRecord	IEC 62055-41 IDRecord (is printable ASCII)
11	required	string	record2	IEC 62055-51 Record2 (with ETX/STX without LRC) (is printable ASCII)
20	required	bool	rollover	The value of the Rollover (RO) bit in the Key Change Token set

4.11 Token (struct)

Any API function that returns a meter-specific STS token will use a Token struct to represent the token and associated metadata. The metadata includes public fields of the Application Protocol Data Unit (APDU; see [IEC 62055-41]), plus token-specific informational fields that may be printed on the receipt.

ID	Presence	Type	Name	Description
1	required	string	drn	DecoderReferenceNumber; 11- or 13-digit DRN, or 18-digit PAN, with valid Luhn(s).
2	required	string	pan	Meter PrimaryAccountNumer; 18-digit PAN, which valid Luhns.

ID	Presence	Type	Name	Description
3	**required**	**i16**	**ea**	EncryptionAlgorithm, range 0-99
4	**required**	**i16**	**tct**	TokenCarrierType, range 0-99
5	**required**	**i32**	**sgc**	SupplyGroupCode, range 0-999999
6	**required**	**i16**	**krn**	KeyRevisionNumber, range 1-9
7	**required**	**i16**	**ti**	TariffIndex, range 0-99
10	**required**	**i16**	**tokenClass**	TokenClass, range 0-3
11	**required**	**i16**	**subclass**	TokenSubClass, range 0-15
12	**required**	**i32**	**tid**	TokenIdentifier, range 0-16777215
13	**required**	**double**	**transferAmount**	TransferAmount (uncompressed); see IEC 623055-41 for range. Value is the actual units represented by the token, which may be rounded (in the customer's favour) from requested amount.
14	**required**	**bool**	**isReservedTid**	True if 'tid' is a SpecialReservedTokenIdentifier, or false for a normal TokenIdentifier.
15	**optional**	**MeterConfigAdvice**	**newConfig**	New meter configuration resulting from a Key Change; field is set if-and-only-if this is the 1st token in a KCT set
20	**required**	**string**	**description**	Human-readable description of the token's purpose (determined by the TokenClass and TokenSubClass), in English.
21	**required**	**string**	**stsUnitName**	Human-readable unit name for transferAmount (English, printable ASCII, short)
22	**required**	**string**	**scaledAmount**	transferAmount scaled to a user-friendly unit and formatted for printing (printable ASCII)
23	**required**	**string**	**scaledUnitName**	Human-readable unit name for scaledAmount (English, printable ASCII, short)
30	**required**	**string**	**tokenDec**	The token as 20 ASCII decimal digits (0-9), for printing or meter card Record1
31	**required**	**string**	**tokenHex**	The token as 17 ASCII hexadecimal digits: 0-9 A-F
40	**required**	**string**	**idSm**	SM identification.STS600-4-2 ID_SM (PKID with rectype SMID) (is printable ASCII)

4.11.1 Special handling for Key Change tokens

No special handling at this time.

4.12 MeterTestToken (struct)

Any API function that returns a non-meter-specific STS token will use a MeterTestToken struct to represent the token and associated metadata. The metadata includes public fields of the Application Protocol Data Unit (APDU; see [IEC 62055-41]), plus token-specific informational fields that may be printed on the receipt.

ID	Presence	Type	Name	Description
1	required	string	drn	DecoderReferenceNumber; 11- or 13-digit DRN, with valid Luhn
2	required	string	pan	Meter PrimaryAccountNumber; 18-digit PAN, with valid Luhns
10	required	i16	tokenClass	Token Class, will be 1
11	required	i16	subclass	TokenSubClass, range 0-15
12	required	i64	control	InitiateMeterTest/DisplayControlField; 28 bits integer for subclass 1,6-10; 36-bit integer for subclass 0,11-15. See See [IEC 62055-41] section 6.2.3 and Table 22
13	required	i16	mfrcode	ManufacturerCode; ranges 0 for subclasses 0,1; 100-9999 for 6-10; 0-99 for 11-15. See [IEC 62055-41] section 6.2.3
20	required	string	description	Human-readable description of the token's purpose (English, printable ASCII). The purpose is determined by the tokenClass and subclass
30	required	string	tokenDec	The token as 20 ASCII decimal digits (0-9), for printing or meter card Record1
31	required	string	tokenHex	The token as 17 ASCII hexadecimal digits: 0-9 A-F

4.13 VerifyResult (struct)

Result returned by method verifyToken.

ID	Presence	Type	Name	Description
1	required	string	validationResult	Result of validation: 'EVerify.Ok' if the token is valid (and exactly one of the optional fields will be present), else an 'EVerify.*' error code from ApiErrors.
2	optional	Token	token	Describes the token when the input was a valid class=0 or class=2 (except KCT) token

ID	Presence	Type	Name	Description
3	**optional**	MeterTestToken	**meterTestToken**	Describes the token when the input was a valid class=1 token

5 Thrift services

The services supported by PrismToken are defined in the Thrift IDL file which is supplied with this document. This section details the purpose of each service.

5.1 TokenApi.ping

The '*ping*' service is used to test connectivity between a Thrift client and server. Calling this service will cause the server to respond with the echo string, after delaying for the sleep period (in milliseconds) specified in the request.

Input arguments (all are required)

ID	Type	Name	Description
1	**i32**	**sleepMs**	The number of milliseconds to delay before responding
2	**string**	**echo**	The string to be echoed back to the client from the server

Output (return) type and exceptions

ID	Type	Description
0	**string**	The function returns the 'echo' string that was supplied in the request.
1	**ApiException**	Thrown if processing fails for any reason.

5.2 TokenApi.signInWithPassword

Signs in with a username and password, to obtain an access token.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**realm**	Authentication realm. Use 'local' to check the username and password against PrismToken's user database.
3	**string**	**username**	Username within the realm; realm & username together identify the subject.
4	**string**	**password**	Cleartext password for the user.
5	**SessionOptions**	**sessionOpts**	Options for this session.

Output (return) type and exceptions

ID	Type	Description
0	**SignInResult**	The result contains the access token.
1	**ApiException**	Thrown if processing fails for any reason.

Comments

- The SignInResult.accessToken is short-lived and must be refreshed periodically. See Access control and authentication (section 3.2) for recommendations on handling accessToken expiry.

5.3 TokenApi.getStatus

Retrieves the status of the PrismToken instance (or a tree of PrismToken instances behind a facade). Each NodeStatus in the output list (min length 1) represents the status of one PrismToken in the tree; the list is ordered as a pre-order depth-first traversal of the tree.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	Bearer token proving authority of the caller; as obtained from a sign-in method

Output (return) type and exceptions

ID	Type	Description
0	list<NodeStatus >	Each NodeStatus in the output list (min length 1) represents the status of one PrismToken in the tree; the list is ordered as a pre-order depth-first traversal of the tree
1	**ApiException**	Thrown if processing fails for any reason.

5.4 TokenApi.parseIdRecord

Parses an IEC 62055-41 IDRecord or IEC 62055-41 Record2 (from a meter card), and returns a MeterConfigIn giving the meter configuration indicated by the record.

IDRECORD is a structure defined in the STS standard (IEC 65044-41) that combines the meter identity (DRN/PAN) and meter configuration (SGC, KRN, TI, EA, TCT). Meter information is not usually handled as an IDRECORD in a production environment; more typically the DRN identifies a database record that stores the meter configuration. This parseIdRecord() method can be helpful in a dev/test environment, in particular because the STS531-* compliance test documents make use of the IDRECORD format.

For reference, the format of an IDRECORD is:

- MeterPAN (18 digits) which is one of:
  - "600727" + 11-digit DRN + CheckDigit
  - "0000" + 13-digit DRN + CheckDigit
  - *DRN is the DecoderReferenceNumber (also called the Meter Serial Number)*
- DateOfExpiry (4 digits) = "0000"
- TokenCarrierType (TCT) (2 digits)
- EncryptionAlgorithm (EA) (2 digits)
- SupplyGroupCode (SGC) (6 digits)
- TariffIndex (TI) (2 digits)
- KeyRevisionNumber (KRN) (1 digit)

“Meter cards” are magstripe cards that can transport a token from a Point Of Sale to a meter; optionally they may store the meter number and configuration (encoded in Record2 on Track3). Meter cards are not widely used; when they are used the POS terminal that reads the card should confirm to an “Entity Type C” (as described in STS531-0, which requires support for IEC 62055-52 and reading the IEC 7813 Record2 structure from the magstripe, all of which is beyond the scope of PrismToken).

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	Bearer token proving authority of the caller; as obtained from a sign-in method
3	**string**	**idRecord**	IEC 62055-41 IDRecord, or IEC 62055-51 Record2 (STX, ETX optional; no LRC) (is printable ASCII)

Output (return) type and exceptions

ID	Type	Description

ID	Type	Description
0	MeterConfigIn	MeterConfigIn giving the meter configuration indicated by the record
1	**ApiException**	Thrown if processing fails for any reason.

5.5 TokenApi.loadTransactionLicense

Uploads the supplied license to the SM such that a non-expired, valid license causes the SM's transaction counter to be set to a non-zero value thereby enabling token vending.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**string**	**licenseText**	String containing the transaction license (including START/END guards); license may be embedded in other text (e.g. an e-mail message).

Output (return) type and exceptions

ID	Type	Description
0	**bool**	If license is successfully processed and the transaction counter is set then returns true.
1	**ApiException**	Thrown if processing fails for any reason.

5.6 TokenApi.issueCreditToken

Issues an STS credit (class=0) token.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**MeterConfigIn**	**meterConfig**	Username within the realm; realm & username together identify the subject.
4	**i16**	**subclass**	Range 0-15
5	**double**	**transferAmount**	Quantity of STS transfer units (the converted amount, not the 16-bit encoded amount).
6	**i64**	**tokenTime**	Effective time of vend, as the number of seconds elapsed since the Unix Epoch (1970/01/01 00:00:00).
7	**i64**	**flags**	Transaction options; bitmap of TokenIssueFlags

Output (return) type and exceptions

ID	Type	Description
0	**list<Token>**	The result contains one or more tokens, with at least one credit token (other tokens may include KCTs, see 7.1 Automatic Key Change Tokens.).
1	**ApiException**	Thrown if processing fails for any reason.

Subclass and units according to IEC 62055-41 Ed 3

Subclass	Resource Type	Resource measure per unit transferAmount	TransferAmount			
			Decimal digits	*Precision*	*Min*	*Max*
0	Electricity	Watt-hours x 100 (0.1 kWh)	0	1	0	18201624
1	Water	0.1 cubic meters				
2	Gas	0.1 cubic meters				
3	Time	0.1 minutes				
4	Electricity currency	10^{-5} base currency	5	0.00001	-1.6383e30	1.6383e30
5	Water currency	10^{-5} base currency				
6	Gas currency	10^{-5} base currency				
7	Time currency	10^{-5} base currency				

Comments

- The range and precision of 'transferAmount' depend on the 'subclass' of the token:
  - For subclasses 0 to 3, 'transferAmount' must be a whole number in the range 0 to 18201624. Decimal digits in the input are ignored (the input is first rounded to 5 decimal places, and then rounded up to the nearest whole number).
  - For subclasses 4 to 7, 'transferAmount' is a floating point number in the range -1.6383e30 to 1.6383e30, with a precision of 0.00001 (5 digits after the decimal point are respected; the input is rounded to 5 decimal places).
  - Subclasses 8 to 15 are reserved by the STS Association.
- Example inputs to create tokens:
  - To create a 100 kWh electricity token, use subclass=0 (electricity) and transferAmount=1000 (1000 units @ 0.1 kWh/unit = 100 kWh).
  - To create a 100 L water token, use subclass=1 (water) and transferAmount=1 (1 unit @ 0.1 m^3/unit = 0.1 m^3 = 100 L).
  - To create a $ 5 USD currency token for a gas currency meter (which has 1 USD as its base currency), use subclass=6 (gas currency) and transferAmount=500000 (500000 units @ 10^{-5} base currency per unit = 5 base currency).
- **Do not** assume that this method only returns 1 token!

5.7 TokenApi.issueMeterTestToken

Issues an STS InitMeterTest (class=1) token (also known as “Non-meter-specific engineering” or “NMSE”).

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method.
3	**i16**	**subclass**	Range 0-15.
4	**i64**	**control**	The InitiateMeterTest/DisplayControlField bitmask, per [IEC 62055-41] section 6.2.3 and Table 22. Range 28 bits for subclass 1,6-10 ; 36-bits for subclass 0,11-15
5	**i16**	**mfrcode**	The MfrCode field, per [IEC 62055-41] section 6.2.3. Range 0 for subclasses 0,1; 100-9999 for subclass 6-10; 0-99 for subclass 11-15

Output (return) type and exceptions

ID	Type	Description
0	**list<MeterTestToken>**	The result contains one or more tokens, with at least one credit token (other tokens may include automatically generated Key Change tokens).
1	**ApiException**	Thrown if processing fails for any reason.

5.8 TokenApi.issueMseToken

Issues an STS management (class=2) token (also known as "Meter-specific engineering" or "MSE").

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**MeterConfigIn**	**meterConfig**	Username within the realm; realm & username together identify the subject.
4	**i16**	**subclass**	Range 0-15
5	**double**	**transferAmount**	Quantity of STS transfer units (the converted amount, not the 16-bit encoded amount).
6	**i64**	**tokenTime**	Effective time of vend, as the number of seconds elapsed since the Unix Epoch (1970/01/01 00:00:00).
7	**i64**	**flags**	Transaction options; bitmap of TokenIssueFlags

Output (return) type and exceptions

ID	Type	Description
0	**list<Token>**	The result contains one or more tokens, with at least one meter-specific management token (other tokens may include KCTs, see See also 7.1 Automatic Key Change Tokens).
1	**ApiException**	Thrown if processing fails for any reason.

Subclass and units according to IEC 62055-41 Ed 3

Subclass	Engineering Token Type	TransferAmount
0	Set Maximum Power Limit	MaximumPowerLimit: the maximum power that the load may draw, in **watts**.
1	Clear Credit	RegisterToClear: the credit register number to clear, or 0xFFFF to clear all credit registers in the payment meter. Credit registers 0 to 7 correspond to the credit token resource types (see issueCreditToken). All other values are reserved by the STS Association.
5	Clear Tamper	Pad: Set to 0.

Subclass	Engineering Token Type	TransferAmount
6	Set Maximum Phase Power Unbalance Limit	MaximumPhasePowerUnbalanceLimit: the maximum allowable power difference between phase loads, in **watts**.

Comments

- Subclasses 2 and 7 are reserved by the STS Association for future definition of the 'Set Tariff Rate' and 'Set Water Meter Factor' tokens.
- Subclasses 3, 4, 8, and 9 are SetDecoderKey tokens used in key change operations, and cannot be generated by this method (use issueKeychangeTokens instead).
- For calculating Power Limit tokens remember that Watts = Volts × Amps.
- Example inputs to create tokens:
    - To create a Set Maximum Power Limit token to limit a 220 Volt supply to 20 Amps, use subclass=0 (Set Maximum Power Limit) and transferAmout= 4400 (220V × 20A = 4400W).
    - To create a Clear Credit token for the electricity credit register only, use subclass=1 and transferAmount=0 (electricity).
    - To create a Clear Credit token that clears all credit registers in the meter, use subclass=1 and transferAmount=0xFFFF (all).
    - To create a Clear Tamper token, use subclass=5 and transferAmount=0.
- **Do not** assume that this method only returns 1 token!

5.9 TokenApi.issueKeyChangeTokens

Issues an STS Key Change Token Set. The result contains the KCT Set which will comprise two or more tokens.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**MeterConfigIn**	**meterConfig**	Username within the realm; realm & username together identify the subject.
4	**MeterConfigAmendment**	**newConfig**	Intended meter configuration after key change. This overrides meterConfig.newConfig, but the toKrn field may be affected by meterConfig.allowKrnUpdate

Output (return) type and exceptions

ID	Type	Description
0	**list<Token>**	The result contains two or more tokens.
1	**ApiException**	Thrown if processing fails for any reason.

5.10 TokenApi.verifyToken

Verifies a previously-issued STS token, returning a verification result (indicates whether the token is valid, or the verification error) and (for a valid token) details of the token's parameters/contents. This method cannot verify a DITK Change Token.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**MeterConfigIn**	**meterConfig**	Username within the realm; realm & username together identify the subject.
4	**string**	**tokenDec**	The token as 20 ASCII decimal digits (0-9)

Output (return) type and exceptions

ID	Type	Description
0	**VerifyResult**	Returns a verification result (indicates whether the token is valid, or the verification error) and (for a valid token) details of the token's parameters/contents
1	**ApiException**	Thrown if processing fails for any reason.

5.11 TokenApi.issueDitkChangeTokens

Issues a Key Change Token Set from a Decoder Initialization Transfer Key (DITK, also known as a Decoder ROM Key) to a Default (VDDK), Unique (VUDK) or Common (VCDK) type Vending Key. This method is only supported if the SM has Manufacturing firmware.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**MeterConfigIn**	**meterConfig**	Username within the realm; realm & username together identify the subject.

Output (return) type and exceptions

ID	Type	Description
0	**list<Token>**	The result contains two tokens.
1	**ApiException**	Thrown if processing fails for any reason.

5.12 TokenApi.fetchTokenResult

Fetches the result of a recent call to a token-issuring method. If a client fails to receive the result of an RPC call, it should use this method to discover the result (if any) or to confirm that PrismToken did not receive the transaction. Simply re-trying a call to a method that issues encrypted tokens will affect the meter's TokenCancellation list and the risk-control counters in the SM.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**string**	**reqMessageId**	Identifies the messageId of the transaction/result to fetch; range IDENT

Output (return) type and exceptions

ID	Type	Description
0	**list<Token>**	The result contains one or more tokens.
1	**ApiException**	Thrown if processing fails for any reason.

5.13 TokenApi.ctsResetTidList

For CTS testing mode only

Zeroises the TokenCancellation lists of zero or more meters. Returns a list of affected DRNs.

Input arguments (all are required)

ID	Type	Name	Description
1	**string**	**messageId**	For logging; use to associate calls with a broader system context.
2	**string**	**accessToken**	As obtained from a signIn* method
3	**string**	**panPattern**	Wildcard match against PANs in the PrismToken database (wildcards: ? and *); range 1-18 digits with optional wildcards (*, ?).

Output (return) type and exceptions

ID	Type	Description
0	**list<string>**	List of affected DRNs
1	**ApiException**	Thrown if processing fails for any reason.

6 Troubleshooting

6.1 C# development

6.1.1 C# certificate is null exception

Modify Thrift libraries to allow non-null certificate ; or use a client-side certificate.

6.1.2 C# AuthenticateAsClient() gives AuthenticationException

In all cases use OpenSSL to check the connection.

SSPI error:

- TLS1.2 may not be supported. Look at the value of System.Net.ServicePointManager.SecurityProtocol immediately before opening the transport and ensure that Tls12 is supported.
    - TLS12 may not be enabled by default on Windows < 8 or .NET Framework < 4.6.2.
    - You can call AuthenticateAsClient(hostname, null, SslProtocols.Tls12, false) to explicitly choose TLS1.2, but MS does not recommend this.
    - Strong Crypto may be disabled in the Windows Registry.
    - See https://docs.microsoft.com/en-us/dotnet/api/system.net.security.sslstream.authenticateasclient?view=netframework-4.7.2
    - See https://stackoverflow.com/questions/35774200/cant-get-sslstream-in-c-sharp-to-accept-tls-1-2-protocol-with-net-framework-4
    - See https://docs.microsoft.com/en-us/dotnet/framework/network-programming/tls MS Docs Transport Layer Security (TLS) best practices with the .NET Framework
    - See MS Docs TLS/SSL Settings https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn786418(v=ws.11)
    - See https://stackoverflow.com/questions/43872575/net-framework-4-6-1-not-defaulting-to-tls-1-2
- Run past the error in the debugger – it may be a low-level exception that is trapped above.

Validation error: RemoteCertificateNameMismatch, etc.

- Regenerate TsmWeb certificate to fix SubjectAlternateName for changed IP address
- Trust the self-signed certificate
- Or use a Validation Callback to accept the certificate despite the above
- Use OpenSSL to look at the certificate

7 Application Notes

7.1 Automatic Key Change Tokens

7.1.1 Caution: automatic Key Change enabled by default

The Automatic Key Change feature is **enabled by default**, so a call to issueCreditToken() or issueMseToken() may return more than 1 token! A fresh Key Load File from the KMC may include new Vending Keys (VKs) or changes to VK attributes that cause PrismToken to start issuing Key Change Tokens (KCTs), which may be surprising if you do not understand this feature.

The STS standard (IEC 62055-41 Ed 3) specifies circumstances in which a meter's key must change (section 6.5.2), and that the vending system may generate the Key Change Tokens (KCTs) manually or automatically. The vending system Compliance Test Specification (STS531-1) includes tests for automatic generation of KCTs (section CTSA19). Ed 1 of the IEC standard *required automatic generation*, which is why this feature is enabled by default in PrismToken.

7.1.2 Caution: Vending System must update meter database

PrismToken does not have a meter database; the Vending System must supply the meter's configuration to every API call that requires it. A Key Change Token changes the meter's configuration, and this change must be captured on the Vending System (or subsequent tokens may be issued using the wrong configuration, and will not work in the meter).

We strongly recommend using the Thrift API (not the Token Issue UI) for all Key Change operations. Although the UI supports key change, it cannot propagate the updated meter configuration to the meter database, and thus should not be used in a Production environment.

7.1.3 Caution: KCTs must be entered into the meter first

Key Change Tokens **must** be entered into the meter first, or the credit token will not be accepted.

7.1.4 Using automatic key change

API methods `issueCreditToken` and `issueMseToken` support Automatic Key Change: in specific circumstances (detailed below) they may issue Key Change Tokens (KCTs) along with the requested Credit/MSE token.

PrismToken will automatically generate KCTs when you call issueCreditToken() or issueMseToken() and:

- The 'meterConfig' parameter includes a 'newConfig' field (type MeterConfigAmendment); OR
- meterConfig.allowKrnUpdate is true (which is the default) and the meter's KeyRevisionNumber (KRN) is not latest KRN for the SGC (the latest KRN is the one with the most recent ActiveDate); OR
- meterConfig.allowKenUpdate is true (which is the default) and the meter's KeyExpiryNumber does not match the KEN attribute of the vending key (identified by SGC and KRN).

Note that the `allowKrnUpdate` and `allowKenUpdate` criteria can be triggered by a change in the Vending Keys (for example by loading a fresh Key Load File from the KMC).

These methods return a `list<Token>`. In typical use only a single token is returned (and it is the requested credit token); but when an automatic key change occurs this list will contain 3 to 5 tokens, with the Key Change Tokens first, then the requested Credit/MSE token (which is protected by the **new** meter key). You must send all these tokens to the meter (via the POS and consumer).

If a KCT is issued then the Token structure (of the first Token in the `list<Token>`) returned by PrismToken will include a `newConfig` field (type MeterConfigAdvice), containing the new configuration of the meter (SGC, KRN, TI, KEN). Your Vending System must store this new configuration to its meter database (or subsequent credit tokens may be issued using the wrong configuration).

7.1.5 Disable automatic Key Change

You can disable automatic KCTs by providing suitable values to the `meterConfig.allowKrnUpdate`, `meterConfig.allowKenUpdate`, and `meterConfig.newConfig` fields.

7.1.6 Recommendations for deployment

We **strongly recommend** that you:

- Consider how you want to manage key changes in your deployed meter base, and choose your `meterConfig` inputs accordingly.
- Test that your system behaves correctly when `issueCreditToken` or `issueMseToken` return more than 1 token.

7.1.7 Prepaid/postpaid mode switching

Some meters support switching between pre-payment and post-payment modes.

Meters compliant with STS202-5

Standards-compliant meters should accept a Class=2 Subclass=10 “SetFlags” token (as defined in STS202-5) to select post-payment (FlagValue=0) or pre-payment (FlagValue=1) for Electricity (FlagIndex=6), Water (FlagIndex=7), Gas (FlagIndex=8), or Time (FlagIndex=9).

PrismToken can generate these tokens:

- Using the API: call the TokenApi.issueMseToken() method with subclass=10 and the transferAmount set to the appropriate Data value (below).
- Using the UI: navigate to the Token Issue page; in the "Generate Tokens" box check "Management Token" and select subclass "10 - Special", then input the appropriate Data value (below).
- Data values for payment modes defined in STS202-5:
    - Electricity Postpaid = 64524
    - Electricity Prepaid = 64525
    - Water Postpaid = 64526
    - Water Prepaid = 64527
    - Gas Postpaid = 64528
    - Gas Prepaid = 64529
    - Time Postpaid = 64530
    - Time Prepaid = 64531

Meters with (non-standard) proprietary payment mode switching

We are aware that some meters use management (class=2) subclass 8 & 9 tokens to select payment mode. **You cannot issue MSE subclass=8 tokens with STS Edition 2 / STS6 hardware**. Although these tokens can be generated by legacy SMs, they cannot be generated by any compliant STS Ed 2 system, as these subclasses have been allocated to key change operations (as explained in the box below).

If you have meters that require subclass=8 or subclass=9 tokens, you will need to discuss options with your meter manufacturer.

Generation of MSE subclass=8 & 9 tokens is not possible with STS Ed 2
In IEC 62055-41 Ed 1 (2007), class=2 (MSE) subclasses 8-10 were declared as "Reserved by STS Association for future assignment". Legacy SMs did not prevent generation of these subclasses, although their meaning was undefined. *Starting around 2013 the STS Association undertook substantial work to address TID rollover and long-term security, with working documents distributed to the membership in 2015 (e.g. STS402-1) for feedback, and subsequently voting. This included the allocation of MSE subclasses 8-9 to Set3rdSectionDecoderKey and Set4thSectionDecoderKey respectively; the Working Group did not receive any objections to this, and the changes were published in IEC 62055-41 Ed 3 (2018).* *The security rules applicable to these subclasses require that* ***the SM can only create them as part of a Key Change operation, and cannot directly create an MSE subclass=8 token****. This applies to all Security Modules compliant with STS Edition 2 and the STS600-8-6 API. So you cannot issue MSE subclass=8 tokens with STS6 hardware.*

8 Amendment History

Version	Description	Person	Date
0.7	Initial draft, corresponds to Thrift file version and emulator prototype.	TD	2016-06-28
0.9	Updated API documentation for Thrift file revision 0.9.	TD, CA	2016-10-25
0.10	Updated API documentation for Thrift file revision 0.10.	TD	2016-12-05
0.10.1	Moved PrismToken overview to "Introduction to STS and PrismToken" (PR-D2-1060)	TD	2018-10-15
1.0	Miscellaneous fixes in preparation for release. Updated template.	TD	2018-10-26
1.0.1	Correction: MeterConfigIn.doe format is 'YYMM' not 'MMYY'.	TD	2019-01-16
1.0.2	Documented supported 'realm' values for signInWithPassword().	TD	2019-01-24
1.0.3	Added 'subclass and units' information tables and comments (including example token inputs) to issueCreditToken() and issueMseToken().	TD	2019-03-07
1.0.4	Added information regarding contents of the info NodeStatus (struct) returned from the TokenApi.getStatus service. Added specification for new service TokenApi.loadTransactionLicense	DP	2020-01-13
1.0.5	Added warning about TLS 1.2 not being enabled by default in .NET Framework < 4.7 and Windows < 8.	TD	2020-04-09
1.0.6	Minor clarifications regarding Windows & .NET support for TLS 1.2.	TD	2020-09-12
1.0.7	Added application note "Automatic Key Change Tokens"	TD	2023-06-12
1.0.8	Clarification on the use cases of parseIdRecord()	TD	2023-11-02
1.0.9	Recommendations on managing longevity and expiry of `accessToken`	TD	2023-12-18
1.0.10	Improvements to application note "Automatic Key Change Tokens"	TD	2024-03-18
1.0.11	Added application note "Prepaid/postpaid mode switching"	TD	2024-07-01