Node2TVA Reference Manual

0.1

Generated by Doxygen 1.4.7

Thu Jan 19 18:45:26 2012

Contents

Chapter 1

Node2TVA How To

1.1 Introduction

Node2TVA allows for node.js scripting to communicate directly with a Tervela appliance stack for publishing and subscribing.

The API for node.js maintains the C style APIs that Tervela releases and maintains. This ensures simple forward compatibility and maintenance.

Node2TVA enables node.js to call Tervela functions directly from node.

1.2 Requirements

1.2.1 Node.js

Latest Node:

- http://nodejs.org/

1.2.2 Tervela Client Lib

Latest tested:

- Windows client library:
  - http://download.tervela.com/ftp/release/R5/5.0_GA/5.0.6_-Composite/clients/Windows/
- Unix client library (tested on Red Hat and Debian):
  - http://download.tervela.com/ftp/release/R5/5.0_GA/5.0.6_-Composite/clients/RHEL/

1.2.3 Node2TVA add-on

Latest Node2TVA add-on (from git):

- https://github.com/Pollenware/Node2TVA

Author:

Todd Weaver <toddw@pollenware.com>

1.3 LICENSE

Chapter 2

Node2TVA Page Index

2.1 Node2TVA Related Pages

Here is a list of all related documentation pages:

Chapter 3

Node2TVA Page Documentation

3.1 Installation

3.1.1 Node.js

- Node.js (example for node-v0.6.1):

```
$ wget http://nodejs.org/dist/v0.6.7/node-v0.6.7.tar.gz
$ tar xvzf node-v0.6.7.tar.gz
$ cd node-v0.6.7/
$ ./configure
$ # NOTE: if configure is missing ssl, try --openssl flags:
$ # ./configure --openssl-includepath=/usr/include/openssl --openssl-libpath=/usr/lib
$ make
$ sudo make install
$ # to run the tests:
$ make test
$ # NOTE: (my node 0.6.1 failed two tests, which I never researched)
```

3.1.2 Tervela Client Lib

- Example for Unix (RH) client library tva-v5.0.6:

```
     $ wget --http-user="<username>" --http-password="<password>" \
       http://download.tervela.com/ftp/release/R5/5.0_GA/5.0.6_Composite\
/clients/RHEL/tva-5.0.6-42984.rhel5_2.6.x86_64.rpm
     $ yum localinstall tva-5.0.6-42984.rhel5_2.6.x86_64.rpm
```

- example for Unix (Debian) client library tva-v5.0.6:

```
$ sudo apt-get install alien
[...]
$ sudo alien tva-5.0.6-42984.rhel5_2.6.x86_64.rpm
Warning: Skipping conversion of scripts in package tva: postinst postrm preinst
Warning: Use the --scripts parameter to include the scripts.
tva_5.0.6-42985_amd64.deb generated
$ sudo dpkg -i tva_5.0.6-42985_amd64.deb
Selecting previously deselected package tva.
(Reading database ... 143476 files and directories currently installed.)
Unpacking tva (from tva_5.0.6-42985_amd64.deb) ...
```

```
Setting up tva (5.0.6-42985) ...
$ #
$ # the follow step is important to dynamically load libtervela at runtime
$ # (this step does not transfer from the rpm to the deb during alien)
$ #
$ sudo su -c "echo '/opt/tervela/lib' > /etc/ld.so.conf.d/tervela.conf"
```

3.1.3 Node2TVA add-on

```
$ git clone git@github.com:Pollenware/Node2TVA.git
$ cd Node2TVA
$ node-waf configure build
```

3.2 Sample Initialize

To use any function it is helpful to have a basic structure, loading the add-on, setting up variables, so here is a quick example:

```
var tervela = require('tervela');

var TVA_OK = 0;

var session = {};
var pub_hndl = {};
var pub_msg_data = {};

var _config = {
    username: "<tmx_username>",
    password: "<tmx_password>",
    tmx: "<tmx_host>",
    num_msgs: 1,
    running: 0,
    sub_type: 113,
};

var t = new tervela.Tervela();

var app = "<app_name>";
var topic = "<TOPIC>";
```

3.3 pwAppInitialize

Example:

```
var rc = t.pwAppInitialize(app);
if (rc != TVA_OK) {
    console.log("pwAppInitialize failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS = pwAppInitialize(appName)
```

DESCRIPTION

Finalizes the configuration for an application and initializes the Tervela API. The function must be called after changing any application parameter setting (pwAppCfgSet) and before any sessions are created (pwSessionNew). This function should only be called once, at the beginning of the application after any configuration is done. The appName parameter is used for monitoring (it can be NULL, in which case the client will use the localhost instead).

PARAMETERS

- appName - char*
  - Input. The name to assign this application (for monitoring).

RETURN VALUE

- TVA_STATUS - If the function succeeds then TVA_OK is returned. If any part of pwAppInitialize fails, it will consider that a Fatal Error and will terminate the application.

ERRORS

- None.

NOTES

- This assigned application name will appear in the client statistic information available from the API and in the TPM Monitoring Network Map view.

3.4 pwSessionNew

Example:

```
function js_notify_cb(c, code, data) {
    console.log("CallBack (%s)", c);
}

rc = t.pwSessionNew(js_notify_cb, _config, session);
if (rc != TVA_OK) {
    console.log("pwSessionNew failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS pwSessionNew(
TVA_SESSION_NOTIFY notifyCb,
void* context,
TVA_SESSION_HANDLE* session);
```

DESCRIPTION

Creates a new session object. This session object will then be used to login to a specific TMX. Multiple session objects can be created and managed by an application.

PARAMETERS

- notifyCb - TVA_SESSION_NOTIFY
  - Input. This callback processes status information for the returned session. You must provide a callback or your application will not succeed at session creation.
  - You should not call any Tervela API functions (except for converting error codes to text, as shown below) or any function that will cause the processor to block (i.e. sleeping or waiting on a mutex) from within your callback function. Instead, set a flag in the callback function and return to the main loop of the application and check for that flag there. The application programmer must ensure that the callback routine executes a minimal amount of code and returns quickly. As a guideline, the callback routine should take no more than 100 ms to execute.
  - (TVA_UINT32 - error code; TVA_STRING - error message string)
  - The prototype for TVA_SESSION_NOTIFY:
  - typedef void (*TVA_SESSION_NOTIFY)(void* context, TVA_STATUS code, void* data);
  - When invoked, the callback notification function will be passed the following arguments:
  - context - The user-defined object (block of memory) that will be returned to the client application when the subscription callback is invoked.
  - code - The event or error code that caused the notification. For example, 151 (TVA_ERR_TMX_FAILED) or 155 (TVA_TMX_SESSION_TERMINATED).
  - data - Any additional information associated with the event, can be NULL (no additional information).
- context - void*
  - Input. Pointer to the user-defined object (block of memory) that will be returned to the client application when the subscription callback is invoked.
- session - TVA_SESSION_HANDLE*
  - Output. The return value. Used to terminate and release all associated resources.

RETURN VALUE

TVA_STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_API_INIT_FAILED
- TVA_ERR_CHANEG_FAILED (63) - The client failed to negotiate a data channel with the TMX. The client will exit on this failure. ("CHANEG_FAILED" is an acronym for (Data) Channel Negotiation Failed.)
- TVA_ERR_MAX_NUM_SESSIONS_REACHED
- TVA_ERR_NULL_ERRORCALLBACK - You didn't supply a callback function. Enter a callback function as described above and re-run your application.
- TVA_ERR_OUTPUT_PARAMETER_NULL

NOTES

- TVA_ERR_API_INIT_FAILED - Returned if the application did not call pwAppInitialize.
- TVA_ERR_MAX_NUM_SESSIONS_REACHED - Returned if the application already has 8 sessions.
- The notifyCb will update your session status. A separate callback routine should be used for each session, if multiple sessions are established. The following describes some of the error and status codes that can be sent as session status. If the client has been disconnected, the client API will attempt to reconnect once a second for 30 seconds before giving up.

3.5 pwSessionLogin

Example:

```
rc = t.pwSessionLogin(session,           // Session handle
                      _config.username,  // API user name
                      _config.password,  // API password
                      _config.tmx,       // Primary TMX
                      0,                 // Secondary TMX
                      0);                // Timeout
if (rc != TVA_OK) {
    console.log("pwSessionLogin failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS pwSessionLogin(
TVA_SESSION_HANDLE session,
const TVA_STRING username,
const TVA_STRING password,
const TVA_STRING primaryTMX,
const TVA_STRING secondaryTMX,
TVA_UINT32 timeout);
```

DESCRIPTION

Takes two TMX names, a primary and a secondary. The secondary can be NULL if the application does not want to log into a fault-tolerant pair.
The client should only set those TMX message switches as primary and secondary that are configured as a fault-tolerant pair in the TPM. If either the TVA_APPCFG_REQUIRE_FT_CONNECT or TVA_SESSCFG_REQUIRE_FT_CONNECT configuration parameter is enabled via pwAppCfgSet / pwSessionCfgSet, a login will only succeed if connections to both TMX message switches were successful. Normally, login will succeed if a connection to at least one of the TMX message switches is successful. If either the primary or secondary login fails (assuming the application uses both), pwSessionLogin will return failure.
An application can maintain multiple sessions, one session per a fault-tolerant pair or a single TMX.
You can have up to 8 sessions per application. There is no restriction on which TMX those sessions connect to but they all need to be in the same namespace (attached to the same TPM), and you need to use the same API User name for each (you will get a TVA_ERR_USERNAME_INVALID error if you do not).
If you are using a fault-tolerant TMX/TPE configuration with GD, your GD publishing / subscribing application must connect to the fault-tolerant TMX pair. See "Using Fault-tolerant Pairs of TMX Message Switches" for information on fault-tolerant pairs of TMX message switches. To establish multiple sessions, the application can call pwSessionLogin multiple times in succession.
Different notification callback routines can be specified for each pwSessionLogin. Each pwSessionLogin call returns a unique session handle and the application must keep track of which session handle belongs to which TMX login. Subsequent API calls that use session handles will have their requests forwarded to the TMX corresponding to the specified session handle. Because pwSessionLogin is not threadsafe, the application must make sure that the previous call has completed before calling it again.

PARAMETERS

- session - TVA_SESSION_HANDLE
  - Input. A handle to the session.
- username - TVA_STRING

  - Input. The API User name defined in the TPM. The TPM must have Topic entitlement rights (publish or subscribe) associated with this API User name before you can publish or subscribe to messages on those Topics.
- password - TVA_STRING
  - Input. The password for the API User in string format. The secure value will be put into the securityContext structure.
- primaryTMX - TVA_STRING
  - Input. The string of either the host name or IP address of the primary TMX. You cannot specify the same TMX (ip address or host name) as both the primary TMX and the secondary TMX when you login.
- secondaryTMX - TVA_STRING
  - Input. The string of either the host name or IP address of the secondary TMX. You cannot specify the same TMX (ip address or host name) as both the primary TMX and the secondary TMX when you login. The secondary TMX is only used if logging into a fault-tolerant pair and can be NULL otherwise.
- timeout - TVA_UINT32
  - Input. The length of time that a login will be attempted before failure. This value is in milliseconds. The system will try to connect to the TMX up to 5 times before failing. Note that under extreme network conditions, the connection can still fail, and return the TVA_ERR_TMX_CONNECTION_FAILED status code. See the NOTES section below for more information on this status code. The login timeout parameter is for the entire logging process: connecting, registering, logging in, and negotiating data channels. Each of these actions occur twice when you log in to a fault-tolerant pair of TMX message switches.

RETURN VALUE

TVA_STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned. The application has access to additional data using pwSessionLogin. The additional data is an array of IP addresses represented as unsigned 32-bit integers (standard IP v4 form). If the connection is fault-tolerant, the first address is the primary TMX and the second address is the secondary TMX. If the connection is to a single TMX, the array is only one element long and contains the address of the connected TMX (which might be useful if the application requested a fault-tolerant connection but only received a single connection to one TMX). The application can use the standard function inet_ntoa() to convert the IP v4 address to a string in standard Internet dotted format.

ERRORS

- TVA_ERR_NETWORK_WRITE
- TVA_ERR_NULL_SESSION_INFORMATION
- TVA_EVT_SEC_TMX_FAILED
- TVA_EVT_SEC_TMX_RECOVERED
- TVA_NOCONNECT_CM
- TVA_ERR_NULL_PASSWORD - You didn't supply a password. Enter a password as described above and re-run your application.
- TVA_ERR_MA_LIST_MISSING - You didn't supply a list of TMXs (formerly called MAs) to connect to. Enter a list of TMXs as described above and re-run your application.
- TVA_ERR_NULL_ERRORCALLBACK - You didn't supply a callback function. Enter a callback function as described above and re-run your application.

- TVA_ERR_NULL_USERNAME - You didn't supply a username. Enter a userName as described above and re-run your application.

NOTES

- Before calling pwSessionLogin, you will need your API User name and password.
- When your application is finished processing all messages, call pwSessionTerm to end the established session.
- Receiving a TVA_ERR_TMX_CONNECTION_FAILED status code means the client could not connect to the TMX. If you get a failure to connect to a TMX, it could be due to temporary network problems. Tervela recommends you retry the login some number of times. Other causes include DNS lookup failure, no network path to the TMX configured, or the TMX is not available.
- The session handle from this call is used to terminate and release all associated resources. Calls that require the session handle from this call are pwCreatePublication, pwSubscribeWithCallback, pwSubscribeWithCallbackEx, and pwSubscribeWithQueue.
- You cannot establish a single session to 2 TMX message switches that are not part of a fault-tolerant Pair. (You can establish multiple sessions to multiple TMX message switches and/or fault-tolerant Pairs.) If you try to log in to 2 TMX message switches that are not configured as a fault-tolerant pair, the following error is returned:
- Login rejected: Invalid fault tolerant configuration

3.6 pwCreatePublication

Example:

```
rc = t.pwCreatePublication(session,       // Session handle
                           topic,         // topicName,
                           0,             // maximumCachePeriod,
                           0,             // enableQOSNotifications,
                           0,             // allowPartialPublish,
                           pub_hndl);     // publisher handle
if (rc != TVA_OK)
{
    console.log("pwCreatePublication failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS pwCreatePublication (
TVA_SESSION_INFORMATION_HANDLE sessionInfo,
const TVA_STRING topicName,
TVA_UINT64 maximumCachePeriod,
TVA_BOOLEAN enableQOSNotifications,
TVA_BOOLEAN allowPartialPublish,
TVA_PUBLISHER_HANDLE *publisher)
```

DESCRIPTION

Registers a publication with a TMX, and stores the handle to the publisher in the publisher parameter. To publish data messages on a Topic, the API User must have the publish entitlements assigned for the Topic in the TPM.
This function is thread-safe; i.e. it never needs to obtain a lock explicitly and can be called concurrently from multiple threads

PARAMETERS

- sessionInfo - TVA_SESSION_INFORMATION_HANDLE
  - Input. A handle to the session for which a publication structure needs to be created. This is created by the pwSessionLogin function.
- topicName - TVA_STRING
  - Input. The string representing the Topic Name. Information published using this publication structure will be published on this Topic.
- maximumCachePeriod - TVA_UINT64
  - Not supported in this release
  - Input. Retransmits between the publisher and TMX are handled out of a publication cache in the publisher, which is sized via the PUB_RATE and PUB_MSG_EXP client.config parameters.
- enableQOSNotifications - TVA_BOOLEAN
  - Not supported in this release. TVA_TRUE must be passed.
  - Input. This enables state and publisher down QOS notifications to be passed to clients consuming the data. Passing TVA_TRUE allows clients to see the status of the publisher.
- allowPartialPublish - TVA_BOOLEAN
  - Not supported in this release.

  - Input. Passing a TVA_TRUE value allows the TMX network to determine if it should allow partial publishing of data values across the network. Partial publishing sends only those fields that have changed, rather than the entire message. This optimizes bandwidth utilization and processing node performance. This functionality is not implemented for this release.
- publisher - TVA_PUBLISHER_HANDLE *
  - Output. It will return the handle to the publisher

RETURN VALUE

TVA _STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_INVALID_TOPIC
- TVA_ERR_NO_RIGHT_PUBLICATION
- TVA_ERR_NO_TOPICNAME
- TVA_ERR_NULL_SESSION_INFORMATION
- TVA_ERR_OUT_OF_MEMORY
- TVA_ERR_OUTPUT_PARAMETER_NULL
- TVA_ERR_SCHEMA_NOT_FOUND
- TVA_ERR_TOPIC_NOT_FOUND
- TVA_TIMEOUT

NOTES

- A Publication structure can be created for a specific or a wildcard Topic. When you send (publish) a message, it must be for a specific Topic. If you register a publication for a specific Topic, then you can simply call the pwCreateMessage function. If you register a publication for a wildcard Topic, you must call the pwCreateMessageForTopic function and specify the Topic to publish on.
- If you create a Publication structure for a wildcard Topic (e.g. AA.BB.*), you can use and reuse it to send multiple specific Topics (e.g. AA.BB.CC and AA.BB.DD and AA.BB.RR) with pwCreateMessageForTopic.
- Use pwCancelPublication when there are no more messages to publish on this Topic.
- TVA_ERR_NO_TOPICNAME is returned when either a null Topic name or a Topic name string with a length of zero is passed to the pwCreatePublication function.
- TVA_ERR_TOPIC_NOT_FOUND is returned when the Topic name cannot be located in the TPM.
- TVA_ERR_SCHEMA_NOT_FOUND is no longer returned by pwCreatePublication although it is returned by other functions.

3.7 pwCreateMessage

Example:

```
rc = t.pwCreateMessage(pub_hndl, pub_msg_data);
if (rc != TVA_OK)
{
    console.log("pwCreateMessage failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS pwCreateMessage(
TVA_PUBLISHER_HANDLE publisher,
TVA_PUBLISH_MESSAGE_DATA_HANDLE *publishMessageData)
```

DESCRIPTION

Returns an empty data buffer that can be loaded with publication data. An existing publishMessage-Data (data buffer) can be reused after publishing when you are using the BE and GC QoS. You shouln't reuse an existing publishMessageData (data buffer) with GD. This function is thread-safe.

PARAMETERS

- publisher - TVA_PUBLISHER_HANDLE
  - Input. The publisher is the handle obtained from a pwCreatePublication call.
- publishMessageData - TVA_PUBLISH_MESSAGE_DATA_HANDLE *
  - Output. This will contain an empty data buffer. Use this data buffer to store all of the fields that you want to publish. When finished using this data buffer, the memory should be released using pwReleasePublishData.

RETURN VALUE

TVA_STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the error codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_API_INIT_FAILED
- TVA_ERR_INVALID_TOPIC
- TVA_ERR_INVALID_TOPIC_FOR_PUBLISHER
- TVA_ERR_MESSAGE_TOPIC_WILDCARD
- TVA_ERR_NULL_PUBLICATION
- TVA_ERR_OUT_OF_MEMORY
- TVA_ERR_OUTPUT_PARAMETER_NULL
- TVA_ERR_SCHEMA_NOT_FOUND
- TVA_WARN

NOTES

- You must call pwCreatePublication before calling pwCreateMessage, which returns the publisher handle. The pointer to this handle is passed to the pwCreateMessage call.

- pwCreateMessage is used when you have registered a publication for a specific Topic. You can create a publication request with a wildcard Topic; you cannot use pwCreateMessage for a wildcard Topic.
- Use pwCreateMessageForTopic when you have registered a publication on a wildcard Topic and need to create a message on a specific Topic.
- If TVA_WARN is returned, the application is running low on allocated resources (buffers). Generally, this is encountered if the application is not freeing buffers when they are no longer needed or it is publishing at too fast of a rate. If it is publishing too fast, either slow down the rate or adjust resources by using the tuning parameters described in the Startup Configuration Parameters section of this document.
- If there are multiple specific Topics (e.g. AA.BB.CC and AA.BB.DD) that will match for a wildcard Topic (e.g. AA.BB.$*$) and you call pwCreatePublication with a wildcard Topic followed by calling pwCreateMessage, you will get the following error: TVA_ERR_MULTIPLE_-SCHEMAS_FOUND.
- publishMessageData is passed to pwSendMessage.
- By default, you will use the default schema assigned to the Topic. If you want to use a different schema, from the same Schema Domain, use pwSelectSchemaForMessage before filling the message with data. Use the pwSet$*$ functions to put data into the fields in the publishMessageData buffer.

3.8 pwSendMessage

Example:

```
rc = t.pwSendMessage(pub_msg_data);
if (rc != TVA_OK) {
    console.log("pwSendMessage failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS pwSendMessage(TVA_PUBLISH_MESSAGE_DATA_HANDLE message)
```

DESCRIPTION

Sends a previously created message to subscribing client applications via a TMX. You must have previously created a message data structure and used the appropriate set functions to store values into the data fields.
This function is thread-safe and can be called concurrently from multiple threads.

PARAMETERS

- message - TVA_PUBLISH_MESSAGE_DATA_HANDLE
  - Input. A handle to the message data structure to be sent/published. This data buffer is initially created with either pwCreateMessage or pwCreateMessageForTopic. It is filled with data for publishing using the pwSet∗ commands.

RETURN VALUE

TVA _STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_API_INIT_FAILED
- TVA_ERR_DATA_PLANE_INACTIVE
- TVA_ERR_FLOW_INACTIVE
- TVA_ERR_FLOW_INVALID
- TVA_ERR_FLOW_NOT_FOUND
- TVA_ERR_INVALID_MESSAGE
- TVA_ERR_MESSAGE_ALREADY_SENT
- TVA_ERR_MESSAGE_DATA_NULL
- TVA_ERR_NO_FIELD_VALUES_IN_PUBMSGDATA
- TVA_ERR_NO_SCHEMA
- TVA_ERR_NULL_PUBLISH_MESSAGE_DATA
- TVA_ERR_OUT_OF_MEMORY
- TVA_WARN

NOTES

- If a certain amount of information is required in the message payload for a specific Topic; it is up to the Subscriber client application to detect if there is sufficient data.

- All Client Applications that have successfully subscribed to this Topic and which have an active session established with the same TMX, or with a TMX with a neighbor route that enables message traffic between these clients, will receive this message.
- pwSendMessage will not return until the message has been sent.
- A flow is maintained for each Specific Topic on which the application is sending messages. The first message that the application sends for a Specific Topic is assigned a Topic Sequence Number (TSN) of 1. The TSN is automatically incremented for each subsequent message on that Specific Topic. When the maximum TSN number is reached, the TSN numbers wrap around to 1 again (0 is skipped). The current maximum for a single Topic is 4,294,967,295. Subscribing applications do not expect a TSN of 0 so a sequence of 0xFFFFFFFF followed by 1 is not considered a gap.
- If TVA_WARN is returned, the application is running low on allocated resources (buffers). Generally, this is encountered if the application is not freeing buffers when they are no longer needed or it is publishing at too fast of a rate. If it is publishing too fast, either slow down the rate or adjust resources by using the tuning parameters described in the Startup Configuration Parameters section of this document.
- Subscribing applications can access the TSN for an individual message through the pwMessage structure.

3.9 pwCreateSchemaFieldIterator

Example:

```
rc = t.pwCreateSchemaFieldIterator(pub_msg_data, field_itr);
if (rc != TVA_OK)
{
    console.log("pwCreateSchemaFieldIterator failed (%d)", rc);
    return rc;
}
```

SYNOPSIS

```
TVA_STATUS pwCreateSchemaFieldIterator
(TVA_MESSAGE_DATA_HANDLE msgDataHandle,
TVA_FIELD_ITERATOR_HANDLE *fieldHandle);
```

DESCRIPTION

Creates an iterator that can be used to loop through the fields in the schema that is used by the given message. That is, each message has a schema associated with it, and this function creates an iterator for that schema. The iterator will iterate over the fields in the schema associated with the message, not the schema in the TPM.

PARAMETERS

- msgDataHandle - TVA_MESSAGE_DATA_HANDLE
  - Input. A handle to the data message for which to retrieve the Schema.
- fieldHandle - TVA_FIELD_ITERATOR_HANDLE*
  - Output. A handle to the schema field iterator

RETURN VALUE

TVA_STATUS - If the function succeeds then TVA_OK is returned. If the function fails, one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_API_INIT_FAILED
- TVA_ERR_INVALID_MESSAGE
- TVA_ERR_MESSAGE_DATA_NULL
- TVA_ERR_OUT_OF_MEMORY
- TVA_ERR_OUTPUT_PARAMETER_NULL
- TVA_ERR_SCHEMA_NOT_FOUND

NOTES

- To get the Schema for a Topic, the API User must be logged in (pwSessionLogin) and must be assigned either the publish or subscribe entitlement for this Topic.
- Once you've obtained the Schema Iterator for a message's schema using pwCreateSchemaFieldIterator, you can use the pwGetNextSchemaField to retrieve the Field ID and Field Type out of the message for each field defined in the Topic's Schema. To get the Field Name from the Field ID, use the pwGetFieldNameFromFieldID function.

- If you already know what the schema is, you do not need to use a field iterator. An iterator should be used when you want to step through every field in the schema and/or get or set every field in the message.
- pwCreateSchemaFieldIterator works with self-describing messages.
- pwReleaseSchemaFieldIterator should be called to free the memory when your application is finished using the iterator.

3.10 pwSchemaFieldNext

Example:

```
rc = t.pwSchemaFieldNext(field_itr, fld_info, i);
if (rc != TVA_OK)
{
    console.log("pwSchemaFieldNext failed (%d)", rc);
    return rc;
}
```

SYNOPSIS

```
TVA_STATUS pwSchemaFieldNext(TVA_FIELD_ITERATOR_HANDLE fieldItrHndl,
TVA_MSG_FIELD_INFO* msgFieldInfo);
```

DESCRIPTION

Returns the schema field information for fields in the message. The function returns the information for one field at a time. It returns a structure that contains two elements, the field ID and the field type. A single structure is reused from the stack, instead of allocating and releasing large numbers of field_-info structures. The field type is an integer that you can then switch on to perform the appropriate type of field operation. This can be for both publishing (Set) and subscribing (Get).
Tervela recommends all new applications use this function as it is faster than using the pwGetNext-SchemaField and pwReleaseMsgFieldInfo combination.

PARAMETERS

- fieldItrHndl - TVA_FIELD_ITERATOR_HANDLE
  - Input. A handle to the field iterator.
- msgFieldInfo - TVA_MSG_FIELD_INFO∗
  - Input. Returned with a structure containing the field information.

RETURN VALUE

TVA _STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_INVALID_ITERATOR
- TVA_ERR_NO_FIELDS_REMAINING
- TVA_ERR_NULL_ITERATOR
- TVA_ERR_OUTPUT_PARAMETER_NULL

NOTES

- None

3.11 pwReleaseSchemaFieldIterator

Example:

```
t.pwReleaseSchemaFieldIterator(field_itr);
```

SYNOPSIS

```
void pwReleaseSchemaFieldIterator (TVA_FIELD_ITERATOR_HANDLE fieldHandle);
```

DESCRIPTION

Releases the memory allocated for an iterator.

PARAMETERS

- fieldHandle - TVA_FIELD_ITERATOR_HANDLE
  - Input. A handle to the schema field iterator.

RETURN VALUE

None

ERRORS

- None

NOTES

- None

3.12 pwReleasePublishData

Example:

```
t.pwReleasePublishData(pub_msg_data);
```

SYNOPSIS

```
void pwReleasePublishData(TVA_PUBLISH_MESSAGE_DATA_HANDLE msgData)
```

DESCRIPTION

Releases the memory allocated for the publish message data structure.

PARAMETERS

- msgData - TVA_PUBLISH_MESSAGE_DATA_HANDLE
  - Input. A handle to the publish message data structure.

RETURN VALUE

None

ERRORS

- None

NOTES

- None

3.13 pwCancelPublication

Example:

```
rc = t.pwCancelPublication(pub_hndl, session);
if (rc != TVA_OK) {
    printf("pwCancelPublication failed (%d)", rc);
}
```

SYNOPSIS

```
TVA_STATUS pwCancelPublication
(TVA_PUBLISHER_HANDLE publicationToCancel,
TVA_SESSION_INFORMATION_HANDLE sessionInfo)
```

DESCRIPTION

Terminates the Publication. All associated Publication objects and data queues are wiped once all data is processed. Any access to those objects might result in null object operations. All pending send (publish) operations will complete.
This function is thread-safe; i.e. it never needs to obtain a lock explicitly and can be called concurrently from multiple threads.

PARAMETERS

- publicationToCancel - TVA_PUBLISHER_HANDLE
  - Input. A handle to the publication to be terminated.
- sessionInfo - TVA_SESSION_INFORMATION_HANDLE
  - Input. A handle to the session information. This parameter is ignored so the function will work even if you don't pass any value (or NULL) for this parameter.

RETURN VALUE

TVA _STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_NULL_PUBLICATION
- TVA_TIMEOUT

NOTES

- To cancel a publication request for a Topic, the API User must have completed a successful call to pwCreatePublication for this Topic.
- After the publication request is cancelled with pwCancelPublication, the API User will not be able to publish any additional messages on this Topic.

3.14 pwSessionTerm

Example:

```
t.pwSessionTerm(session);
```

SYNOPSIS

```
TVA_STATUS pwSessionTerm(TVA_SESSION_HANDLE session)
```

DESCRIPTION

Ends the current session. All associated session objects, publication, subscriptions, and data queues are wiped. Any access to those objects might result in null object operations.

PARAMETERS

- session - TVA_SESSION_HANDLE
  - Input. A handle to the session. Used to terminate and release all associated resources.

RETURN VALUE

TVA_STATUS - If the function succeeds then TVA_OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

ERRORS

- TVA_ERR_INVALID_HANDLE
- TVA_ERR_NULL_SESSION_INFORMATION
- TVA_ERR_NETWORK_WRITE
- TVA_EVT_SESSION_TERMINATED
- TVA_NOCONNECT_CM

NOTES

- All messaging activity should be stopped before terminating the session with pwSessionTerm.
  - Applications that are subscribing to Topics should call pwTerminateSubscription before calling pwSessionTerm.
  - Applications that are publishing on Topics should call pwCancelPublication before calling pwSessionTerm.
- When you start a client session (using pwSessionLogin), you are assigned a unique session ID. All message routing is done based on that session ID. When the client application session is terminated, it is removed from the TMX routing tables and no further messages are routed to it and the application will no longer be able to publish or receive any messages for any Topic.
- Calling pwSessionTerm will clear the session handle. Any attempts to use a session handle after calling pwSessionTerm will result in a TVA_ERR_INVALID_HANDLE (13) error. If a session is closed asynchronously by the API (slow consumer, reconnect failed, etc.), further attempts to use the session handle by the application will result in a TVA_EVT_SESSION_TERMINATED (155) status code.