Conformance Test Suite (CTS_2.1.3.32.0)

Installation

Install prerequisites

- Please uninstall VTS before proceeding with the CTS installation.

  ```
  sudo pip uninstall Vts
  ```

- On Ubuntu systems:

  ```
  sudo apt-get -y update
  ```

  ```
  sudo apt-get -y install python-pip python-setuptools git
  python-lxml python-dev python-pysqlite2 build-essential
  libssl-dev libffi-dev
  ```

- On RedHat based systems:

  ```
  dnf update -y
  ```

  ```
  dnf install -y python-pip python-setuptools git python-lxml
  python-devel redhat-rpm-config
  ```

- Python dependencies:

  ```
  sudo pip install "lxml>=3.5.0" "bs4>=0.0.1" "configparser>=3.5.0"
  "requests>=2.9.1" "tabulate>=0.7.5" "sqlalchemy" "simplejson>=3.8.1"
  "rstr>=2.2.4" "colorama>=0.3.7" "jsonpointer>=1.9" "pyopenssl"
  "ndg-httpsclient>=0.4.0" "pyasn1>=0.1.9" "backports.functools_lru_cache>=1.3"
  "pandas>=0.19.2"
  ```

  ```
  sudo pip install "Flask-Bootstrap==3.3.7.1" --no-dependencies
  ```

Installation from pre-built .tar.gz archive

1. Remove old /opt/cts data

   ```
   [sudo] rm -rf /opt/cts
   ```

2. Download tar.gz package with CTS

3. Extract tar.gz package:

   ```
   tar -zxvf CTS_PACKAGE.tar.gz
   ```

4. Enter extracted directory:

   ```
   cd CTS_PACKAGE
   ```

5. Install cts wheel package (pip requires correctly configured connection to public repositories: proxy, routes, etc)

   ```
   [sudo] pip install CTS*.whl
   ```

6. Copy the configuration.ini file to location /etc/cts/configuration.ini:

   ```
   [sudo] mkdir -p /etc/cts/
   [sudo] cp configuration.ini /etc/cts/
   ```

7. Copy test scripts to /opt/cts/tests:

   ```
   [sudo] mkdir -p /opt/cts/tests/
   [sudo] cp -r tests_packages/* /opt/cts/tests/
   ```

8. Create directory for logs:

   ```
   [sudo] mkdir -p /var/log/cts
   ```

9. If CTS files have been created using a root account, their ownership has to be changed to the user that will be executing CTS:

   ```
   [sudo] chown -R USER_NAME /opt/cts/
   [sudo] chown -R USER_NAME /var/log/cts
   ```

Configuration

- CTS framework configuration
  - Open file /etc/cts/configuration.ini
  - Following flags can be edited:
    - **SQL_CONNECTION_STRING** - defines database to be used by cts
      set flag following instructions http://docs.sqlalchemy.org/en/latest/core/engines.html (can require installing additional packages)

- Test configuration:

  CTS requires that test configuration is part of every `cts execute` command. It is possible to use a single configuration file, but CTS accepts more than one configuration file and the user may find it convenient. If used in this way, configuration files are concatenated and in case of conflict, value from file to the right takes precedence over value from file to the left.

  - create config_file.ini with keys:
    - **ApiEndpoint** - endpoint to API in format ip:port
    - **UseSSL** - defines if CTS shall use http or https protocol to connect to api (Yes/No)
    - **CertificateCertFile** and **CertificateKeyFile** - paths to client side pem certificate and key files (if API requires client certificate authorization)
    - **User** and **Password** - User and Password used by CTS to authorize (if API requires basic authorization)

The above set of keys is mandatory for all tests and tells the CTS how to connect to API endpoint. Other optional keys are:

  * **IgnoreTypes** - User may may specify that entities of certain types should not be validated by CTS. Use comma to ignore more than one type. Example: `IgnoreTypes=Bios.1.1.0.Bios, TypeToIgnore.1.0.1.TypeToIgnore`
  * **MapTypes** - User may declare that CTS should use known type to validate specific unknown type. Example: `MapTypes=ComputerSystem.v2_0_0:ComputerSystem.v1_0_0` CTS will use ComputerSystem.v1_0_0 schema do validate ComputerSystem.v2_0_0 entities
  * **ServiceTypeOverride** - If your service implements more than one service type (e.g. combined PSME and RMM), you have to inform CTS that entities form both services may exist together on the REST. This will prevent CTS from raising errors about unknown RMM types when PSME test package is executed. Possible values are: PODM_2_1, PSME_2_1, RMM_2_1, SS_2_1 Example: `ServiceTypeOverride=PSME_2_1,RMM_2_1`

– create hardware_check_list.ini with configuration required by hardware_check_list test script. You may skip this file if checking hardware requirements is not planned. You may also add these keys to config_file.ini created above if you prefer to maintain a single configuration file.

  * **IsPsmePresent** - Please declare (Yes/No). Rack must have one or more Pooled System Management Engine software (PSME)
  * **PowerEfficiency** - declared power efficiency (number < 100)
  * **FanPositionNumberingConsistent** - Please declare (Yes/No). Service personnel should be able to easily identify the location of a failed fan for replacement. It is recommended that the fan position location label use the base as 1 and be numbered from left to right or right to left or top to bottom or bottom to top, within each subsystem (rack, drawer, or module).
  * **PowerSupplyPositionNumberingConsistent** - Please declare (Yes/No). Service personnel should be able to easily identify the location of a power supply failure for replacement. It is recommended that the power supply position location label use the base as 1 and be numbered from left to right or top to bottom, within each subsystem (rack, drawer, or module).
  * **IsRMMNetworkPrivateManagement** - (Yes/No)
  * **AreManagementAndProductionNetworksSeparated** - (Yes/No)
  * **AreComputeBladesServiceableIndependently** - (Yes/No)

Basic Usage

Browsing available test packages, test suites and test cases

- To list all available test packages:

  ```
  cts tests list
  ```

- To filter by package name:

  ```
  cts tests list -p Rack_Scale_2_1_POD_Manager
  ```

- To filter by package and test suite names:

  ```
  cts tests list -p Rack_Scale_2_1_POD_Manager -s required
  ```

- To generate a sample configuration file for test case:

  ```
  cts tests generate_config PACKAGE_NAME TEST_SCRIPT_NAME \
  -o output_file_with_configuration.ini
  ```

Execution

- To simply execute all tests for Rack_Scale_2_1_POD_Manager validation:

  ```
  cts execute tests Rack_Scale_2_1_POD_Manager \
  --config_files config_file.ini
  ```

- To execute only tests for metadata compliance:

  ```
  cts execute tests Rack_Scale_2_1_POD_Manager --test_suites
  required \
  --config_files config_file.ini
  ```

- To execute only a test validating get responses' compliance with provided metadata:

  ```
  cts execute tests Rack_Scale_2_1_POD_Manager --test_scripts
  validate_get_responses \
  --config_files config_file.ini
  ```

- To set timeout for each script executed, add flag -T –timeout"

  ```
  cts execute tests Rack_Scale_2_1_POD_Manager --config_files
  config_file.ini \
  -T timeout_in_seconds
  ```

Test results browsing

- To list all test executions:

  ```
  cts status list
  ```

- To show detailed information about a specified execution:

  ```
  cts status show RUN_ID
  ```

- To delete a result from the CTS database:

  ```
  cts status delete RUN_ID
  ```

- To save results to a file (when you choose *html* option, in your work-directory will be created a new folder *cts-reports* which will include *html* files):

  ```
  cts status dump RUN_ID --output_format [html/csv/text]
  ```

Additional options

- Show CTS Version:

  ```
  cts version
  ```

Advanced Usage

Using run list to execute multiple tests with a single command

Run list is a mechanism that enables multi-step execution and makes it possible to plan more advanced test scenarios. Execution of a run list is very similar to executing a test script.

```
cts execute run_list run_list_2_1
```

`run_list_2_1` is a test specification prepared by the user. It defines scope of tests to be executed as well as configuration that should be used to run tests. Below is an example of a run list that can be used to execute all 2.1 tests:

```
$ cat run_list_2_1

[PSME_2_1]
TEST_PACKAGE = Rack_Scale_2_1_PSME
TEST_SUITES = required
TEST_CONFIGS = ./config/psme.ini, ./config/hardware_check_list.ini

[StorageServices_2_1]
```

```
TEST_PACKAGE = Rack_Scale_2_1_Storage_Services
TEST_SUITES = required
TEST_CONFIGS = ./config/storage.ini

[PODM_2_1]
TEST_PACKAGE = Rack_Scale_2_1_POD_Manager
TEST_SUITES = required
TEST_CONFIGS = ./config/podm.ini, ./config/hardware_check_list.ini
```

The run list definition refers to additional configuration files:

```
 $ cat config/psme.ini
[PSME]
ApiEndpoint = <IP:PORT>
UseSSL = Yes
User = admin
Password = admin

$ cat config/storage.ini
[StorageServices]
ApiEndpoint = <IP:PORT>
UseSSL = Yes
User = admin
Password = admin

$ cat config/podm.ini
[PSME]
ApiEndpoint = <IP:PORT>
UseSSL = Yes
User = admin
Password = admin

$ cat config/hardware_check_list.ini
[HardwareCheckList]
PowerEfficiency = 90
FanPositionNumberingConsistent = Yes
PowerSupplyPositionNumberingConsistent = Yes
IsRMMNetworkPrivateManagement = Yes
AreManagementAndProductionNetworksSeparated = Yes
```

Test Description

API Get Validation

You can run the API Get Validation tests by passing a flag:

```
--test_scripts validate_get_responses
```

The test is read-only checking that resources exposed on the REST service are compliant with metadata being part of the RSD specification. CTS will raise an error if any of the following conditions occur:

- property defined as 'required' is not present
- unknown property is detected in an instance of type that does not allow additional properties
- value of a property has incorrect type
- value of a property has a value out of range (resulting from type itself or when min/max values are defined)
- value of a property does not match a defined regular expression (if any are defined in metadata)
- resource or object of an unknown type is detected

API Get Validation is the most basic test that is available in CTS and as such should be executed as the first test since API correctness is a prerequisite for the rest of the tests.

API Patch Validation

You can run the API Patch Validation tests by passing a flag:

```
--test_scripts validate_patch_responses
```

The test iterates through all resources discovered on REST API in search of patchable properties (i.e. properties with OData.Permissions/ReadWrite annotation declared in metadata). Based on the property definition, CTS generates a new value and issues a PATCH request followed by GET for verification purposes. If return codes and verification results are correct, the test case passes. Otherwise, CTS reports a warning. When CTS finishes with an actual API resource, the original state is restored.

We usually advise running this test in the next order after the API Get Validation.

Hardware Checklist Validation

You can run the Hardware Checklist Validation tests by passing a flag:

```
--test_scripts hardware_check_list
```

The test suite consists of both manual and automatic tests that verify that requirements from the Platform Design Specification document are met.

Manual tests require an additional set of configuration parameters to work - please refer to the Test Configuration section. CTS verifies that values provided by the user are conformant with PDS.

The scope of automatic testing is as follows:

- CTS verifies that at least one compute module is present in the POD
- CTS verifies that at least one Ethernet switch is present in the POD
- CTS verifies that at least one Ethernet-based fabric is present in the POD
- CTS verifies that API endpoint uses secure channel (SSL) to serve API
- CTS checks if client is able to perform computer system reset
- CTS checks that all API chassis resources define Location
- CTS checks that chassis hierarchy is consistent
- CTS checks that Power Monitoring is possible at Rack level

CRUD tests

You can run CRUD (Create Read Update Delete) tests by passing a flag:

```
--test_scripts crud_operations
```

The test tries to create an instance of a resource, then checks if it was created correctly. After that, it attempts to patch the resource, checks the correctness again and finally deletes it. The test is supposed to clean up the changes it made no matter when it comes to a stop (e.g. if it created resources incorrectly, we skip the patch but still try to delete it). The following resources are tested:

- PSME: VLAN network interface (without patching - not supported by the REST API)
- Storage Services: Logical Drive, Remote Target

Troubleshooting

- I ran a test (with the specified *–test_scripts*), but nothing happened.

  ‘ $ cts execute tests Rack_Scale_2_1_POD_Manager –test_scripts validate_patch_responses -c config_files.ini

  Using CTS in version 2.2.14.0 No scripts where selected to execution ‘

Probably, CTS files in `/opt/cts/` have been created using a root account and their ownership has to be changed:

```
`[sudo] chown -R USER_NAME /opt/cts/`
```

- I got an error `IOError`, when I ran a test.

Probably, CTS directory `/var/logs/cts/` has been created using a root account and its ownership has to be changed:

```
`[sudo] chown -R USER_NAME /var/log/cts`
```

Known issues

```
HSD111437   : CTS (PSME): PATCHing null on DriveErased returns Bad request
Problem     : CTS patch test is based on metadata, which specifies NULL as
              one of permitted values.
              Metadata does not contain information that a null value
              for this property is possible only as a temporary
              (when correct value is undetermined) value. PSME returns error
              because it cannot accept null as a new value.
              To fix it, CTS test needs to be redesigned and rewritten.
              Quick fix is not possible here and the change is planned
              in the future.
Implication : CTS log contains false positive WARNING after if tries to
              patch DriveErased with Null value.
Note        : CTS
Workaround  : No workaround
Exposure    : Medium
```

```
HSD116710   : CTS does not detect that EventService on PSME
              is not metadata-compliant
Problem     : CTS does not detect illegal properties whose names start
              with a # (hash) sign.
Implication : CTS may not report an error when an illegal property
              starts with a #.
Note        : CTS
Workaround  : No workaround
Exposure    : Low
```