CCSDK-2871 DP: Long-term Java API(s)

Introduction

Although the PoC will only implement a few of the possible Java API methods it is important to have a good detailed view of the structure and naming of this interface going forward and document it.

Acceptance Criteria for Proposed Java Interface:

1. Should follow ONAP or wider best practice
2. Documented on ONAP Wiki
3. Discussed and agreed within CPS Team
4. Discussed and agreed with wider community

Currently we are considering 3 'separated' Java APIs or 'groups' of methods:

1. Models (add, list)
2. Data (CRUD)
3. Queries

Jira Ticket:

CCSDK-2871 - Getting issue details... STATUS

Gerrit Review

https://gerrit.nordix.org/#/c/onap/ccsdk/features/+/6477/

External Resources

https://wiki.onap.org/display/DW/Data+Representation

https://wiki.onap.org/display/DW/Interface+style

Open Issues/Decisions

#	Description	Details	Decisions
1	Should the java interface take in one (JSON) objects (like REST interface) or a few individual fields in a signature?	• for DataNode attributes (Fragments) in our DB we will use a 'glorified' Map to store these and pass them on through the layers as a single object • For API methods that require 2-4 parameters there is no need to create separate objects	• Payload will be glorified map • For non payload parameters, if there is more than 3/4 they might get encapsulated in an object. (to be decided on a case by case basis)
2	Input streams and/or files to take in large amounts of data like yang models?	• Input streams are more generic • Overloading of API to support both leads bloated interface	03/11/20 Team meeting Niamh, Toine, Rishi, Aditya, Bruno, Phillipe We have decided to use (buffered) input streams.

3	API uses (generated) ID's or customer provided names? If names are used should we return IDs at all?	03/11/20 Team meeting notes - Niamh, Toine, Rishi, Aditya, Bruno, Phillipe Pros: 1. Using IDs would have some performance advantage 2. Using the ID seems natural in some cases 3. Using IDs would allow for 'renaming' where possible Cons: 1. Using ID would mean client has to get/cache ID's all the time. 2. Supporting both names and ID will lead to 'duplicate' methods in Java API and REST API 3. Using a meaningless ID for Module instead for namespace & revision could lead to issues when debugging (only logs with IDs available) Other considerations: 1. The Java API follow the REST API. If the rest API is using names then the java API should also use names. 2. Create methods should return the (id of the) objects created. If the module already exists, an exception should be thrown. 3. All ID's generated should be in the response. If we return the ID we also need methods to use the ID. If we update the java API to use ID, we should also update the REST API. 4. OSS RC does NOT use IDs for models 5. OSS RC Does DB Generated ID for objects (fragments) 6. Need separate decision for a. Dataspace b. Module Set c. Module d. Anchor e. Fragment	04/11/20 Team meeting notes - Niamh, Toine, Rishi, Tony We should not expose internal DB ID for dataspace, module set, module, anchor or fragment. DB ids should be fully encapsulated. Exposure of DB ids to clients limits freedom in the future. We do not want to expose them for the following reasons: • Database Migration Database technology uplift/swap out will not guarantee that ids will be immutable. Any client that uses DB ids would be broken. I can see this being quite important for ONAP trails going to commercially supported ONAP deployments. • Database Optimization We would like to reserve the right to change database ID's. You eliminate the option that will shuffle the ID's without impacting the client. • ID Persistence If there is a fragment that has restricted properties that can only be set at creation. This fragment will be identified by an xpath and internally by a DB id. An update of this (restricted) attribute will require a delete /create – this will change the DB id, but not the xpath. If a client had cached (or otherwise stored) the DB id, then their use case would be broken 1. Dataspace - Name will be provided by user 2. Module Set -Name will be provided by user 3. Module - Name will be provided by user 4. Anchor - can have multiple ID's (will be mapped to entities outside of CPS). We will return URI for an anchor. We will use a generic key value pair map. We will do a check to make sure it is unique and return URI for anchor. 5. Fragment - XPath is the correct unique identifier and will be added automatically
4	Should a user be able to delete a dataspace, module ((of the same revision), module set?	1. Maybe only allow if no associated data exist e.g. all fragments using a module should be deleted before a module can be deleted. 2. If the associated data still exists, an exception could be thrown (from REST API)	04/11/20Team meeting notes - Niamh, Toine, Rishi, Tony • We want to able to delete an anchor • We should be able to delete a dataspace - un deploying use case • We need to be able to delete the module and module sets associated with the dataspace. • Rest API (or any other possible user interface) will have safety checks before deletion: ◦ Dataspace - do not delete if any module(set) s are still associated ◦ Module Set - do not deleted if any modules (or possible anchors) associated ◦ Module - do not delete if any any anchor associated with it ◦ Anchor - delete allowed from REST (will delete everything underneath) ◦ Fragment - delete allowed from REST (will delete everything underneath)
5	Should a user be able to update/override (create again) dataspace, module (of the same revision), module set?	1. Can add business logic to check on create if it already exists. If it exists (and is the same) we do not create it (silently ignore, this is called 'idempotent'.	04/11/20 Team meeting notes - Niamh, Toine, Rishi, Tony For now we are not going to be idempotent but we may consider it in the future. Need to be document clearly as part of Java API proposal (i.e in this wiki!)

6	How should we specify attributes?	1) can it be done by xpaths 2) additional parameters 3) part of query builder?	11/11/20 Team meeting notes - Niamh, Toine, Aditya, Tony, Ruslan, Claudio We will use an optional parameter (suggested name outputSpecification) to specify the leaf(s) (attributes). All queries will return DataNode(s). If you do not specify leaf(s), all leaves will be returned.
7	Should we have one method with any type of paths to handle both direct gets with a fully qualified paths(pointing to one unique DataNode) and any type of more advanced paths expression/querys (pointing to multiple DataNodes)? Theoretically this is possible but if might be confusing for the end user	alternative we could have more specialized methods like getDataNode(String cpsPath) queryDataNodes(String cpsPathQuery) And we could still consider an path builder pattern (like a query builder) PathBuilder.withChildOfType() PathBuilder.withAttributeValue()	11/11/20 Team meeting notes - Niamh, Toine, Aditya, Tony, Ruslan, Claudio We decided to use key instead of ID. We have decided to split the behavior into different methods e.g getDataNode(String cpsPath) queryDataNodes(String cpsPathQuery) We do intend to use a query builder too and it will be extended in step with any progress on the xpath query supported functionality
8	Will we have separate methods to validate our modules or data? Alternatively validate on submitting of data/modules (and throw validation exception if needed)		11/11/20 Team meeting notes - Niamh, Toine, Aditya, Tony, Ruslan, Claudio We intend to publish validation in a separate java library. We will do validation on write to ensure whatever is being stored in the database is valid and will give the user a fast failure. We will not have separate validation methods on the main Java API.

9	The parser has detailed exception classes instead of one exception with different messages, we need to ask ourselves if we want to have a similar pattern	See section on error handling.	11/11/20 Team meeting notes - Niamh, Toine, Aditya, Tony, Ruslan, Claudio
			
			We intend to have our own detailed validation list of validation exception classes.
			
			We will have an exception class hierarchy like below:
			
			• CPSValidationException
			  ◦ ModelValidationException
			  ▪ SpecificException1
			  ▪ SpecificException2
			  ◦ DataValidationException
			  ▪ SpecificException3
			  ▪ SpecificException4
			  ◦ PathValidationException
			  ▪ SpecificException5

10	Order of parameters		When applicable the following order should be applied 1. Dataspace 2. ModuleSet 3. Module 4. Namespace 5. Revision 6. Anchor 7. CpsPath
11	Responsibility of module sets will be clarified in a meeting	Options are: 1. Client (owner of the modules) 2. CPS	• CPS will be responsible for storing module sets. • Module sets are stored as source definition. • The client will be responsible for providing a unique name within a dataspace. • The module set uniqueness within a dataspace to be identified by string identifier now, also by content later

Error Handling

Exception Class Hierarchy:

- CPSValidationException
  - ModelValidationException
    - SpecificException1
    - SpecificException2
  - DataValidationException

        - SpecificException3
        - SpecificException4
    - PathValidationException
        - SpecificException5

#	Class	Definition	Source	Proposed Exception Class
1	YangValidationException	Unchecked exception thrown if yang definition is not valid according to {YangModelBasicValidationListener}	ODL Yang Parser (Yang Model)	YangValidationException
2	YangParseException	Unchecked exception thrown if unable to parse yang	ODL Yang Parser (Yang Model)	YangValidationException
3	YangSyntaxErrorException	Exception thrown if yang syntax is invalid	ODL Yang Parser (Yang Model)	YangValidationException
4	DataValidationException	Exception thrown when a the specified data is invalid	ODL Yang Parser (Yang Data)	YangDataValidationException
5	LeafRefDataValidationFailedException		ODL Yang Parser (Yang Data)	YangDataValidationException
6	LeafRefPathSyntaxErrorException		ODL Yang Parser (Yang Data)	YangDataValidationException
7	ModifiedNodeDoesNotExistException	Exception thrown when a proposed change fails validation before being applied into the Data Tree because tree node which child nodes are modified or written did not exist when transaction started and still does not exists when transaction is processed. Note if node existed in first place and was removed by other transaction, thrown exception should be ConflictingModificationAppliedException.	ODL Yang Parser (Yang Data)	NodeDoesNotExistException
8	ConflictingModificationAppliedException	Exception thrown when a proposed change fails validation before being applied into the Data Tree because the Data Tree has been modified in way that a conflicting node is present.	ODL Yang Parser (Yang Data)	DataModificationException
9	MissingSchemaSourceException	Exception thrown when a the specified schema source is not available.	ODL Yang Parser (Yang Model)	MissingSchemaSourceException
10	SchemaResolutionException	Exception thrown when a Schema Source fails to resolve.	ODL Yang Parser (Yang Model)	SchemaSourceException
11	SchemaValidationFailedException		ODL Yang Parser (Yang Model)	SchemaValidationException
12	OperationFailedException	A general base exception for an operation failure.	?	

13	AlreadyExistsException		Database duplicates	ModelAlreadyExistsException DataAlreadyExistsException
14	IO Exception			N/A

Interface Proposal

can be found at : Interface Proposal for CPS

Proposed grouping of interface methods:

Interface Name	Interface Capabilities
ModuleStoreService	• String createModuleSet(String dataspaceName, String moduleSetName, InputStream modulesData); • Collection<ModuleRef> getModuleReferences(String dataspaceName); • Void deleteModuleSet(String dataspaceName, String moduleSetName);
`CpsAdminService`	• String createDataspace(String dataspaceName); • Collection<String> getDataspaces(); • void deleteDataspace(String dataspaceName); • String createAnchor(String dataspaceName, String anchorName); • Collection<Anchor> getAnchors(String dataspaceName); • Anchor getAnchor(String dataspaceName, String anchorName); • void deleteAnchor(String dataspaceName, String anchorName); • void associateAnchorToModuleSet(String dataspaceName, String moduleSetName, String anchorName);
`DataService`	• String addDataNode(String dataspaceName, String anchorName, DataNode dataNode); • String addDataNode(String dataspaceName, String anchorName, String xPath, DataNode dataNode); • Integer count(String dataspaceName, String anchorName, String cpsPath); • Integer count(String dataspaceName, String cpsPath); • void updateDataNode(String dataspaceName, String anchorName, DataNode dataNode); • void setLeaf(String dataspaceName, String anchorName, String xPath, String leafName, Object leafValue);
~~`QueryService`~~	• ~~DataNode getDataNode(String dataspaceName, String anchorName, String xPathId);~~ • ~~Collection<DataNode> getDataNodes(String dataspaceName, String xPathId);~~ • ~~Collection<DataNode> queryDataNodesByCPsPath(String dataspaceName, String cpsPath);~~ • ~~Collection<DataNode> queryDataNodesBySchemaNodeIdentifier(String dataspaceName, String schemaNodeIdentifier);~~

Java Doc

JavaDoc.zip