String API

\*\*\* 2.2\*\*\*

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Major Changes |
| 1.0 | 02-Mar-2014 | [chris@daxcloud.com](mailto:chris@daxcloud.com) | Initial draft for Phase 1 |
| 2.0 | 18-Jul-2014 | [Venkat.g@primefocus.com](mailto:Venkat.g@primefocus.com) | Updated APIs based on the discussion with Chris |
| 2.1 | 21-Oct-2014 | [Raghavendra.s@primefocus.com](mailto:Raghavendra.s@primefocus.com)  [Kiran.ummarasetty@primefocus.com](mailto:Kiran.ummarasetty@primefocus.com) | Document updated with sample service input and output |
| 2.2 | 38-Oct-2014 | [Raghavendra.s@primefocus.com](mailto:Raghavendra.s@primefocus.com) | Architecture diagram updated |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Data Model (For Phase 1)

### Users

|  |  |
| --- | --- |
| id: int (autogen unique) | 7 |
| password: string | Password123 |
| userGUID: string | (128-bit integer) |
| email: string | [chris@daxcloud.com](mailto:chris@daxcloud.com) |
| isAdmin:Boolean | 0 or 1 |
| Username:string | Chris |
| Oldpassword:string | old@123 |

128-bit UUID is normally represented by unsigned char data[16]

User are created by ETC or company admins. Users are not tied with assetIDs

### Assets

|  |  |
| --- | --- |
| id: int (autogen unique) | 23 |
| assetHash: String | 0d6dce44a124443d845af191c3d1b64 |
| serviceToken: String | 64 (length) |
| URI: string | [https://my.service.com/lookup/blah?asset=%assetHash%](https://my.service.com/lookup/blah?asset=%25assetHash%25) |
| description: string | The parent owner of this asset and all of its derivatives is the USC ETC. |
| metadata :string | As of now only description only. |
|  |  |

The asset URI points to the meta-data of the asset.

### Service

|  |  |
| --- | --- |
| id: int (autogen unique) | 5 |
| serviceName: string | Ownership Information |
| serviceToken: string | OWNER |
| isPublic : Boolean | True |
| serviceDescription: string | This service type provides ownership information about the asset. A non-parameter call to the URI will provide more information on how to access this service. |

### Usersession

|  |  |
| --- | --- |
| id: int (autogen unique) | 5 |
| userName: string | Raghu |
| userSession: string | Fklsjhflsdkahfakjhskfhkejrhe |
| sessionTime: Timestamp | 2014-10-21 18:20:08 |
| userGuid: string | oiudfoisdufoiioeuroisdufoiwuanlkajdluia |

# API

The following is a basic set of services to call the String metadata API. While the HTTP call will most likely use a JSON POST, this remains to be architected. We have included an example of the HTTP calli in the login API call. In addition, the return format will need to be determined as well.

## Login/Identify User

This is used to login to the String API for the first time. After Phase 1, we will use the Keystone authentication API. It returns a session token to be used for the remaining calls.

Input

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/Login

input:

{

"username":"admin",

"password":"admin"

}

output:

{

result: "54v35v1fmnbr1boj8gu6uiov3ph3k25b"

errorMessage: null

}

## Change Password

Necessary to change the password. Reset will allow for a new password to be set and emailed to the user’s email address.

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/ChangePassword

input

{

"oldPassword":"kiran",

"newPassword":"Kiran@78",

"reEnterPassword":"Kiran@78",

"userSession":"my session",

"username":"myname2",

"userSession":"7tu6heh9ha3i31mimvqdl6t63ok23bs0"

}

output

{

result: {

id: 20

username: "mahesh"

password: "Mahesh@1234"

oldPassword: "Mahesh@123"

userGuid: "s4hemv60lfpm1h8c2vdiqfu9a81mkm0bdocjpedcuiucto2gs4jes8s6eudbm3mn"

email: "mahesh@primefocus.com"

isAdmin: false

userSession: null

}-

errorMessage: null

}

## Register Asset ID with Service

The primary registration method to associate an asset ID with a service type and a URI for more information. The call takes an optional description for this association to be retrieved as part of the lookup. In addition, publicly available metadata key-value pairs can be registered through the optional metadataPairs structure, which is a JSON-encoded, square-bracketed, quoted list of data.

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/AddAsset

input:

{

"assetHash":"e5adfc1f5c3171d145c584d2b9b91d721as",

"url":"https://www.asset.com/location",

"description":"Testing desc",

"metadata":"metadata changed",

"userSession":"54v35v1fmnbr1boj8gu6uiov3ph3k25b",

"serviceToken":"fctj637cgvr3h2uilh14h3beqfl6a4p7lh0rhqs55geeds8fmarlmqt98n87lrsi"

}

output:

{

result: "e5adfc1f5c3171d145c584d2b9b91d721as"

errorMessage: null

}

## Update AssetID Service (Phase 2)

This API allow admins to update the assetID information. This API is open to only admins

Not yet implemented

## Delete AssetID Service (Phase 2)

This API allow admins to delete an existing assetID. This API is open to only admins

Not yet implemented

## Register User

To register a user or service provider into the system, provide the userID, password, and contact email. If successful, the system will return the assigned userGUID.

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/AddUser

ADD operation

input

{

"password":"Mithun@123",

"email":"mahesh@primefocus.com",

"isAdmin":0,

"username":"Mithun",

"userSession":"54v35v1fmnbr1boj8gu6uiov3ph3k25b"

}

output

{

result: "User added successfully"

errorMessage: null

}

## Register Service

To register a service type, provide the human-readable name for the service, a token value, and a longer description. The service ID will be autogenerated.

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/AddService

input

{

"serviceName":"My Upload",

"serviceDescription":"Upload description",

"userSession":"54v35v1fmnbr1boj8gu6uiov3ph3k25b"

}

output

{

result: {

id: 5

serviceName: "My Upload"

serviceToken: "j53e6h4rnprnretqpst0vaq9opmvmnh5q95cmtmt9lmkn52iha3aq2l1bd4deqsa"

isPublic: null

serviceDescription: "Upload description"

userSession: "54v35v1fmnbr1boj8gu6uiov3ph3k25b"

}-

errorMessage: null

}

Services are registered by ETC or company admins

## Query Registry

The Query service takes asset hash and user token as input and returns the details of the asset.

<https://maid.host.com/queryAsset?hash=0d6dce44a124443d845af191c3d1b645&userToken=789d569f08ed7055e94b4289a4195012&ServiceToken=64pns1238botqohqskjpoeurqjn95iqibqv4guadm4gtngm5a0sgcq410krga3re>

Response:

{

{

“service”:”OWNER”,

“ URI”:” [https://my.service.com/lookup/blah?asset=%assetHash%](https://my.service.com/lookup/blah?asset=%25assetHash%25)”,

“description”:”The parent owner of this asset and all of its derivatives is the USC ETC.”,

“metadata”:

{

“Original”:”True”,

“Color-corrected”:“False”,

“Copyright”:“2014 USC ETC”

}

}

}

## GetAllServicesByAssetHash

This API shall return all available services in the system for a given AssetID.

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/GetAllServicesByAssetHash

input:

{

"userSession":"54v35v1fmnbr1boj8gu6uiov3ph3k25b",

"assetHash": "assethash-1"

}

output:

{

result: [1]

0: {

id: 3

serviceName: "View"

serviceToken: "64pns1238botqohqskjpoeurqjn95iqibqv4guadm4gtngm5a0sgcq410krga3re"

isPublic: 1

serviceDescription: "View description"

userSession: null

}-

-

errorMessage: null

}

## GetServices

This api returns all the services which are public

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/GetServices

input:

{

"userSession":"54v35v1fmnbr1boj8gu6uiov3ph3k25b",

"isPublic":1

}

output:

{

result: [4]

0: {

id: 2

serviceName: "jack updated"

serviceToken: "sdfshhkjh121k2h12kjh12k1h2"

isPublic: 1

serviceDescription: "jack description"

userSession: null

}-

1: {

id: 3

serviceName: "View"

serviceToken: "64pns1238botqohqskjpoeurqjn95iqibqv4guadm4gtngm5a0sgcq410krga3re"

isPublic: 1

serviceDescription: "View description"

userSession: null

}-

2: {

id: 4

serviceName: "Download"

serviceToken: "9ni7ppfhrhoggcou22a3ledof8n5v3aboamvli4hkmhcd3357uj35a75ce6vtl04"

isPublic: 1

serviceDescription: "Download description"

userSession: null

}-

3: {

id: 5

serviceName: "my upload updated"

serviceToken: "sdfshhkjh121k2h12kjh12k1h2wwewe"

isPublic: 1

serviceDescription: "my upload description"

userSession: null

}-

-

errorMessage: null

}

## GetAsset

This api returns asset details

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/GetAsset

input:

{

"userSession":"e6l80c7c7gf1b68alkhmjtu2uu10qdfo",

"assetHash":"assetHash-1"

}

output:

{

result: [2]

0: {

id: 5

assetHash: "assethash-1"

url: "https://asset.com/location"

description: "Testing"

metadata: "metadata changed"

serviceToken: ""

userSession: null

}

1: [2]

0: "fctj637cgvr3h2uilh14h3beqfl6a4p7lh0rhqs55geeds8fmarlmqt98n87lrsi"

1: "64pns1238botqohqskjpoeurqjn95iqibqv4guadm4gtngm5a0sgcq410krga3re"

errorMessage: null

}

## UpdateAsset

This api updates an asset with new details

<http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/UpdateAsset>

input:

{

"id":"2",

"assetHash":"assethash-1",

"url":"new url",

"description":"new desc",

"metadata":"new metadata ",

"userSession":"e6l80c7c7gf1b68alkhmjtu2uu10qdfo",

"serviceToken":"fctj637cgvr3h2uilh14h3beqfl6a4p7lh0rhqs55geeds8fmarlmqt98n87lrsi"

}

output:

{

result: "true"

errorMessage: null

}

## Add Parent/Peer Registry (Phase 2)

## Configure Synchronization (Phase 2)

## Logout

http://devbms05.dev.daxplatform.com:38080/StringAPI/Service/Logout

input:

{

"userSession":"lkua8cv3ev8asgtamj0fruoi35hal376"

}

output:

{

result: "User Logged out Successfully"

errorMessage: null

}

STRING: DEFINITION

String is name of a unique identifier for an asset.  Using the Sha-2 256 Secure Hash Algorithm, a unique identifier or thumbprint of the asset is created based upon the file itself.  Since the string is created based upon the file, any time the hash process is run against the file, the same unique identifier is produced assuring that the asset is the same.

String provides a registry service allowing the unique identifier to be registered and used to allow collaboration and sharing of data around the asset being secure in the knowledge that asset is the correct asset.

Once the asset is registered in the String Registry, participating organizations can add services to the asset that extend the types of data and actions that can be taken on the asset.  The extension of the services are managed by the String Administration Group who verify candidate services the deploy them.

Here is a scenario that explains how this works:

1) A DIT is capturing Metadata on set.

2) He runs a tool to get the String for a video file and captures the string number (hash)

3) The DIT then writes down the metadata and saves the metadata to a file.

4) The video file is sent to a pre-production company to create Dailies

5) At the pre-production company, they upload the asset into their Production Management Software

6) Their PM system runs a tool to get the String for a video file and logs into the String system and registers the asset String.

7) 2 days later, the DIT logs into his companies Prod Management tool to upload the metadata.

8) Upon the upload of the metadata the system queried if the DIT has a String

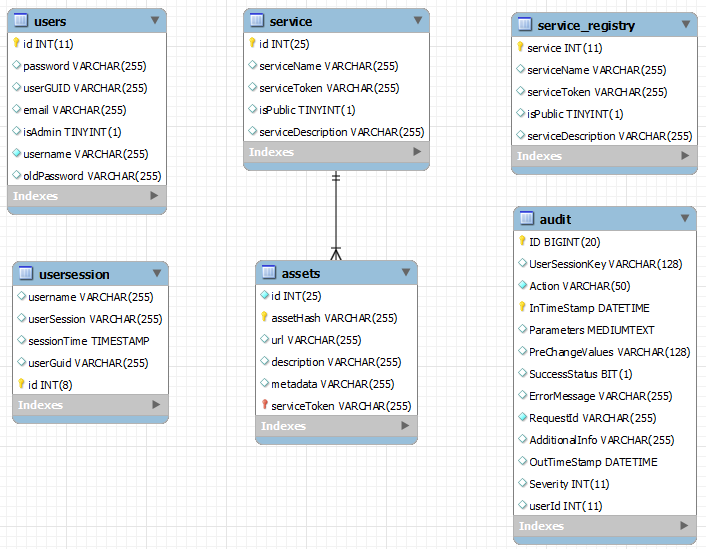
9) The DIT entered the string.

10) The system checks the string registry and sees that there is already a registered string so it adds a new service URL for the metadata into the asset's string registry.

11) 3 days later, a user at the pre-production company accesses the video and the system checks to see if there is metadata associated with the asset in the String Registry.

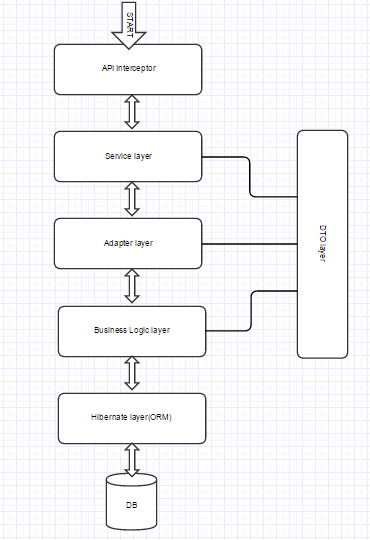
12) It sees that there is metadata and calls the service, gets the metadata, and displays it on the screen to the end user.

## DB Snapshot



## Architecture Diagram

Below diagram shows the architectural snap shot of STRING rest API’s. Detailed layers in the STRING project



**Interceptor**

Interceptor(PreProcessInterceptor,PostProcessInterceptor, MessageBodyReaderInterceptor ) are used to capture audit details for each services.

Per-ProcessInterceptor: This runs after a resource method is found to invoke on, but before the actual invocation happens. Generating random String and set object. This string is used as unique key for logging audit. In this interceptor we inserting (In-timestamp, remote details, method name, RequestID(random String)) into DB.

Post-Process Interceptor: This will run after the MessageBodyWriter and after the ClientRequest has been totally built on the client side. This interceptor is used to capture the services is passed or fail. If fails the error message is captured in exceptionMapper.

MessageBody-ReaderInterceptor : Interceptor is used to capture the Inputs of the service for audit log.

**Service layer**

Service layer is exposed to outside world. The service interface defines the operations that service supports and their associated parameters and data transfer objects.

This will consumes and produces JSON string.

**Adapter layer**

This class contains logic of validating inputs from services. Adapter will use converters to convert the BDO (Business Data transfer Objects) to DTO (Data transfer Objects) to avoid expose details of the business entities.

**Business** **layer**

This is the layer all the business logic for th api exist. It used the BDO to set get into the database through ORM layer.

**ORM layer**

It will communicate with database through ORM. Current application we are using hibernate as the ORM tool