Workspace_Deluxe Documentation

Release TBD

Gavin Price

Dec 06, 2017

CONTENTS

CHAPTER

ONE

OVERVIEW

The Workspace Service (WSS) is a language independent remote storage and retrieval system for KBase typed objects (TO) defined with the KBase Interface Description Language (KIDL). It has the following primary features:

- Immutable storage of TOs with
  - user defined metadata
  - data provenance
- Versioning of TOs
- Referencing from TO to TO
- Typechecking of all saved objects against a KIDL specification
- Collecting typed objects into a workspace
- Sharing workspaces with specific KBase users or the world
- Freezing and publishing workspaces
- Serverside extraction of portions of an object

CHAPTER

TWO

MANUAL

2.1 Workspace fundamentals

The workspace service (WSS) provides language independent storage and retrieval of typed objects (TOs) much like those from various object oriented programming languages (except, of course, no behavior is associated with WSS TOs). Before an object is stored in the WSS, it is checked against a type specification that describes the object's structure and contents. Failing this check will abort the storage operation.

2.1.1 Typed object relationships

TOs can be related to each other in various ways (see Figure 1):

Workspaces

Workspaces are arbitrary (from the perspective of the WSS) disjoint collections of TOs. Any meaning given to the collection is provided by the users creating that collection. For instance, a workspace might collect TOs representing known microbial genomes as a public resource, or, for a single genome, reads, an assembly, a genome, and a metabolic model that represent a single researcher's work on a microbe.

Each TOs is in one, and only one, workspace.

Workspaces may be shared with other users, allowing read, write, or administration privileges. These privileges apply to all TOs in the workspace.

Workspaces have a user-selected name and an integer ID that is immutable and assigned on creation. See *Workspaces* for more details.

Versions

TOs saved in the WSS are immutable. Saving 'over' a TO, rather than replacing it, creates a new version of the object. Most of an object's properties may differ from version to version, but the object's user-selected name, immutable integer ID (again assigned at creation) and a few other states are the same for all versions. See *Objects* for more details.

References

The WSS supports two types of references: depencency references and provenance references.

A dependency reference is a reference that implies an object is dependent on another object to function - a Genome on a ContigSet, for example. Dependency references are embedded in the object itself and are called out in the type

specification (see *Workspace typed objects*). They can thus be required, if desired, and an object without such a dependency reference will fail to save.

In contrast, a provenance reference implies that an object was produced from another object. These are not called out in the type specification and are not embedded in the object.

Establishing data provenance is required for usable data and repeatable science. Without provenance data for a data object, said object might as well have been made from the whole cloth. Reproducing the data is impossible, and it is impossible to judge the data's reliability.

An application or user needs the object referred to in a dependency reference to compute on the referencing object; they do not need any provenance references. A dependent object may or may not be part of the referring object's provenance - for example a Genome and ContigSet could be produced at the same time from a GenBank file and so the ContigSet would not be part of the Genome's provenance. Rather, they would share the same provenance.

When creating an object version, dependency and provenance references can be specified, thus recording which other objects are required to use that version of the object, and which other objects are required to understand the creation of that object.

Both types of references have another special property - they guarantee access to the referent, regardless of permissions or deletion state, as long as the user has access to the referring object. The philosophy behind this permanent access is that a data object is useless without its provenance and dependencies as described above.

See *Objects* for more information on storing objects with references, and *Traversing object references* for information on accessing referents.

Copies

Copying a TO from another TO implies that the new TO is effectively identical to the old TO - it possesses the same data and references. It may not have the same name or ID or exist in the same workspace. Unlike references, access to a copied object does not provide any special rights to the copy source.

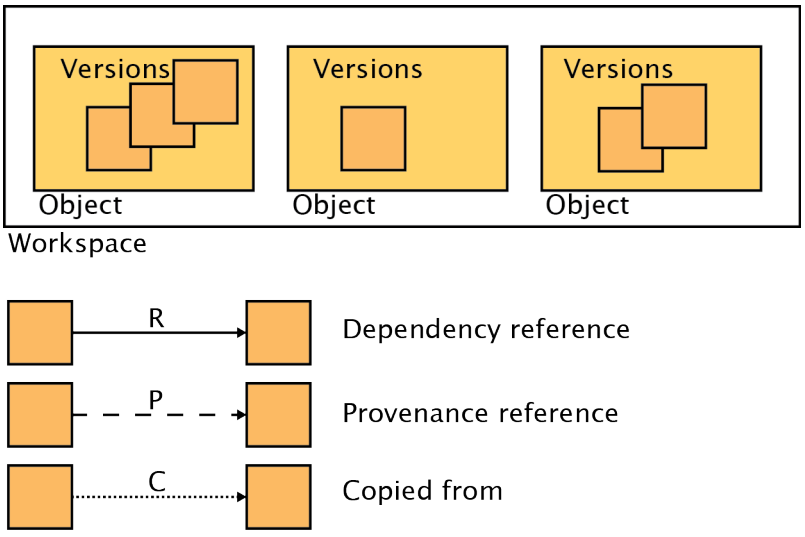

Fig. 2.1: Figure 1: Types of relationships between workspace service objects

2.1.2 Addressing workspaces and objects

Workspaces may be addressed by either their mutable name or permanent ID.

Objects may be addressed by a combination of the workspace name or ID, the object's mutable name or permanent ID, and optionally a version in the reference format `[workspace name or ID]/[object name or ID]/[version]`.

As an example, assume that an object with name `MyObj`, ID 12, and 3 versions exists in a workspace with name `MyWS` and ID 4. The following are all valid addresses:

Address	Targets version
MyWs/MyObj	3
MyWs/12/2	2
4/MyObj	3
4/12/1	1
4/12/3	3
4/12	3

2.1.3 The object graph

The various relationships between objects create a graph structure of nodes (object versions) connected by edges (versions, references, and copies). Specifically, the objects form a directed acylic graph (DAG). As previously described, in the case of references the DAG may be traversed without limit *in the forward direction*, e.g. from referencing object to referent, starting with an object to which the user has direct access. A user may also traverse the DAG in the reverse direction, but only to objects to which the user already has direct access. See *Traversing object references* for more details. It is possible to traverse the DAG from copy to copy source, but again only if the copy source is directly accessible by the user.

2.1.4 Example

Figure 2 provides an example of how an object graph might look after a few operations.

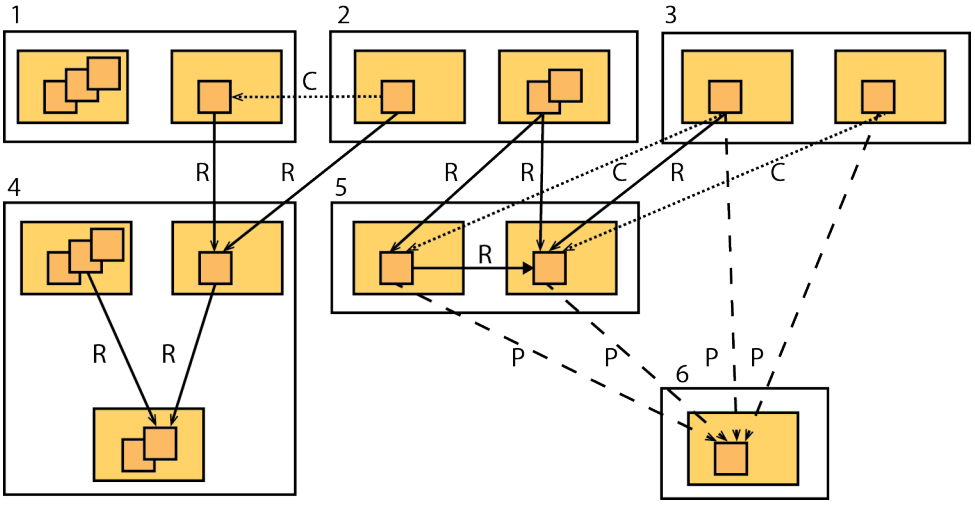

Fig. 2.2: Figure 2: An example object graph

Assume that the objects in each workspace are numbered, starting at 1 in the upper left corner and incrementing for each object as one moves along the row. At the end of a row, the object on the next row receives the next number and the process continues.

Workspace #4 has three objects. Object 3 has two versions, neither of which have outgoing references. Object 1 has 3 versions. Version 2 of object 1 has a dependency reference to version 2 of object 3. Object 2 has one version which has a dependency reference to version 2 of object 3.

Workspace #1 has two objects. Object 1 has three versions, none of which have outgoing references. Object 2 has a single version with a reference to the single version of object 2 in workspace 4 - e.g. `4/2/1`.

Thus, as described above, a user with access to workspace #1 also has access to the objects addressed by `4/2/1` and `4/3/2` via object 2.

A user with access to workspace #4 has no access to object `1/2/1` unless explicitly granted such by an administrator of workspace #1 (which would allow access to all objects in workspace #1).

Workspace #6 has a single object with a single version with no outgoing references. Although it has four incoming references, they provide no privileges for the referencing objects.

Workspace #5 has two objects with one version each. Object 1 has a provenance reference to the object in workspace #6 and a dependency reference on object 2 in the same workspace. Object 2 has the same provenance reference as object 1.

Workspace #2 has two objects. Object 1 has a single version that was copied from object `1/2/1`. Object 2 has two versions, the first of which has dependency references to both objects in workspace #5.

Since object 1 was copied from object `1/2/1`, which has a dependency reference to object `4/2/1`, object 1 has the same reference and the same access to workspace #4s objects as object `1/2/1`.

If the user examining object 1 also has access to workspace #1, the information that object 1 was copied from object `1/2/1` will be available. If not, the user will know the object was copied, but not from where.

Since object 2 has two outgoing dependency references as described, access to object 2 also provides access to objects `5/1/1`, `5/2/1`, and `6/1/1`.

Workspace #3 was cloned from workspace #5 (theoretically this should be impossible since the workspace with the lower ID must have been created first, but for the purposes of this example ignore that). It has the same two objects as workspace 5, and those objects possess the same references as the objects in workspace 5. In particular, object 1 has a dependency reference to object `5/2/1` (just as object `5/1/1` does) and both objects possess provenance references to the object in workspace #6. Both objects also have copy references to their source objects in workspace #5, but again, these references provide no special privileges.

2.2 User documentation

2.2.1 Build documentation

This documentation assumes the documentation build occurs on Ubuntu 12.04LTS, but things should work similarly on other distributions. It does **not** assume that the KBase runtime or `dev_container` are installed.

Requirements

The build requires:

Java JDK 8+

Java ant:

```
sudo apt-get install ant
```

Python Sphinx 1.3+:

```
curl https://bootstrap.pypa.io/get-pip.py > get-pip.py
sudo python get-pip.py
sudo pip install sphinx --upgrade
```

Getting the code

Clone the jars and workspace_deluxe repos:

```
bareubuntu@bu:~/ws$ git clone https://github.com/kbase/jars
Cloning into 'jars'...
remote: Counting objects: 1466, done.
remote: Total 1466 (delta 0), reused 0 (delta 0), pack-reused 1466
Receiving objects: 100% (1466/1466), 59.43 MiB | 2.43 MiB/s, done.
Resolving deltas: 100% (626/626), done.

bareubuntu@bu:~/ws$ git clone https://github.com/kbase/workspace_deluxe
Cloning into 'workspace_deluxe'...
remote: Counting objects: 22004, done.
remote: Compressing objects: 100% (82/82), done.
remote: Total 22004 (delta 41), reused 0 (delta 0), pack-reused 21921
Receiving objects: 100% (22004/22004), 21.44 MiB | 2.44 MiB/s, done.
Resolving deltas: 100% (14000/14000), done.
```

Build

Build the documentation:

```
bareubuntu@bu:~/ws$ cd workspace_deluxe/
bareubuntu@bu:~/ws/workspace_deluxe$ make build-docs
```

The build directory is `docs`.

2.2.2 Build and initialize the workspace client

This documentation describes how to build and initialize the workspace clients. It assumes the documentation build occurs on Ubuntu 12.04LTS, but things should work similarly on other distributions. It assumes that the `workspace_deluxe` and `jars` repos have been cloned (see *Getting the code*) but does **not** assume that the KBase runtime or `dev_container` are installed.

Python client

Currently the Python client only supports Python 2.7. The Python client checked into `libs/biokbase/workspace/client.py` does not require a build, but does require the `requests` (v 2+) 3rd party library, which, depending on the Python version, can be tricky to install securely. The following incantation worked for the author:

```
sudo apt-get install python-dev libffi-dev libssl-dev
curl https://bootstrap.pypa.io/get-pip.py > get-pip.py
sudo python get-pip.py
sudo pip install --upgrade requests
sudo pip install --upgrade requests[security]
```

For python 2.7.9+ `sudo pip install --upgrade requests` should work.

Change the working directory to the lib directory:

```
bareubuntu@bu:~/ws$ cd workspace_deluxe/lib/
bareubuntu@bu:~/ws/workspace_deluxe/lib$
```

Alternatively, add this directory to the `PYTHONPATH`. If deploying with the `dev_container`, the client will be copied to `/kb/deployment/lib/biokbase/workspace/client.py` and the `user-env` script will set up the `PYTHONPATH`.

Here we use the iPython interpreter to demonstrate initializing the client, but the standard python interpreter will also work:

```
bareubuntu@bu:~/ws/workspace_deluxe/lib$ ipython
```

```
In [1]: from biokbase.workspace.client import Workspace
In [2]: ws = Workspace('https://kbase.us/services/ws', user_id='kbasetest',␣
 ↪password=[redacted])
In [3]: ws.ver()
Out[3]: u'0.3.5'
```

Java client

The Java client build requires:

Java JDK 6+ (install instructions)

Java ant:

```
sudo apt-get install ant
```

Build the client:

```
bareubuntu@bu:~/ws/workspace_deluxe$ make compile-java-client
ant compile_client
Buildfile: /home/bareubuntu/ws/workspace_deluxe/build.xml

compile_client:
    [mkdir] Created dir: /home/bareubuntu/ws/workspace_deluxe/client_classes
    [javac] Compiling 48 source files to /home/bareubuntu/ws/workspace_deluxe/client_
 ↪classes
      [jar] Building jar: /home/bareubuntu/ws/workspace_deluxe/dist/client/
 ↪WorkspaceClient.jar
   [delete] Deleting directory /home/bareubuntu/ws/workspace_deluxe/client_classes

BUILD SUCCESSFUL
Total time: 3 seconds
```

The client jar is created in `dist/client/WorkspaceClient.jar`.

For simplicity, copy the required jars into a single directory:

```
bareubuntu@bu:~/ws$ mkdir tryjavaclient
bareubuntu@bu:~/ws$ cd tryjavaclient/
bareubuntu@bu:~/ws/tryjavaclient$ cp ../workspace_deluxe/dist/client/WorkspaceClient.
 ↪jar .
bareubuntu@bu:~/ws/tryjavaclient$ cp ../jars/lib/jars/jackson/jackson-annotations-2.5.
 ↪4.jar .
```

```
bareubuntu@bu:~/ws/tryjavaclient$ cp ../jars/lib/jars/jackson/jackson-core-2.5.4.jar .
bareubuntu@bu:~/ws/tryjavaclient$ cp ../jars/lib/jars/jackson/jackson-databind-2.5.4.
↪jar .
bareubuntu@bu:~/ws/tryjavaclient$ cp ../jars/lib/jars/kbase/auth/kbase-auth-0.4.4.jar␣
↪.
bareubuntu@bu:~/ws/tryjavaclient$ cp ../jars/lib/jars/kbase/common/kbase-common-0.0.
↪24.jar .
bareubuntu@bu:~/ws/tryjavaclient$ ls

jackson-annotations-2.5.4.jar        kbase-auth-0.4.4.jar
jackson-core-2.5.4.jar               kbase-common-0.0.24.jar
jackson-databind-2.5.4.jar           WorkspaceClient.jar
```

When creating an application using the WSS it's advisable to use a build tool like `ant`, `maven`, or `gradle` to organize the required jars.

This simple program initializes and calls a method on the WSS client:

```
bareubuntu@bu:~/ws/tryjavaclient$ cat TryWorkspaceClient.java
```

```
import java.net.URL;
import us.kbase.auth.AuthConfig;
import us.kbase.workspace.WorkspaceClient;
import us.kbase.auth.ConfigurableAuthService;
import us.kbase.auth.AuthToken;

public class TryWorkspaceClient {

    public static void main(String[] args) throws Exception {
        String authUrl =
            "https://ci.kbase.us/services/auth/api/legacy/KBase/Sessions/Login/";

        ConfigurableAuthService authService = new ConfigurableAuthService(
                new AuthConfig().withKBaseAuthServerURL(new URL(authUrl));

        String tokenString = YOUR_AUTH_TOKEN_HERE;
        AuthToken token = authService.validateToken(tokenString);

        WorkspaceClient client = new WorkspaceClient(
                new URL("https://ci.kbase.us/services/ws/"),
                token);
        System.out.println(client.ver());
    }
}
```

Compile and run:

```
bareubuntu@bu:~/ws/tryjavaclient$ javac -cp "./*" TryWorkspaceClient.java
bareubuntu@bu:~/ws/tryjavaclient$ java -cp "./:./*" TryWorkspaceClient
0.8.0
```

For more client initialization and configuration options, see *Workspace API documentation*.

Perl client

Todo: Build and initialization instructions for the Perl client. If this can be done without the KBase runtime & dev_container that'd be ideal.

Javascript client

Todo: Build (probably not needed) and initialization instructions for the Javascript client.

2.2.3 Workspace typed objects

The Workspace Service (WSS) provides storage, sharing, versioning, validation and provenance tracking of typed object (TO) data. This document describes basic information for developers who need to define and register TOs for use with the WSS.

Typed object basics

TOs in the WSS are hierarchical data objects that conform to type definitions specified in the KBase Interface Description Language (KIDL). Just as KIDL is used to specify the structure of data exchanged between KBase clients and servers (generated by the Type Compiler), KIDL is used to define the structure of data stored in the WSS.

Any structures defined in a KIDL formatted file (e.g. `typedef structure { ... } StructureName;`) can be registered with the WSS (see *Typed object registration & versioning*). Instances of these objects can then be saved to the WSS by any user. The WSS does not support storage of primitive or basic container types directly (ie string, int, float, list, mapping).

KIDL defined Modules provides namespacing for typed objects. Thus, the module name and type name is required to uniquely identify a type in the WSS, generally in the format `ModuleName.TypeName`.

Typed object registration & versioning

TO definitions must be registered with the WSS before instances of the TOs can be saved. The basic process for registering a TO is:

- Developer requests ownership of a module name via the Workspace API
  - see API method `request_module_ownership(...)`
- WSS admin approves the request
- Developer uploads (i.e. registers) a type specification file (typespec) in KIDL format where the module name is identical to the just approved module name in the WSS and indicates the names of the TOs which the developer wants the WSS to support
  - see API method `register_typespec(...)`
- Developer releases the module, which releases the latest version of all TO definitions in the module
  - see API method `release_module(...)`

TO definitions marked for WSS usage are versioned with a major and minor version. Every time a new typespec is uploaded and registered, the TO definitions defined in the module automatically receive a new version number if changed. Minor versions are incremented if the change is backwards compatible (i.e. addition of a new optional field). Major versions are incremented if the change is not backwards compatible.

All versions of all registered TO definitions are available to WSS users, but to save an object instance of an old version, or an unreleased version, the exact version number must be provided by the user. If a WSS user saves an object instance without providing version numbers for the type, the latest released version of the TO definition is assumed. The process of releasing a module therefore indicates that the latest version of all typed object definitions in the module are ready for public use, but does not limit user's or developer's ability to work with old or pre-released versions of TOs.

Before the first release of a module, repeated uploads of a module result in version numbers of TO definitions of 0.x and are assumed to be backwards incompatible. On first release of a module, all version numbers of TO definitions are updated to 1.0.

Users and developers can use the `ws-typespec-list` script or the API to list registered modules, type definitions, and versions of type definitions, and to retrieve the actual KIDL or JSON schema encoding of the typed object definition. End users will only be able to view the versions of TOs that are released. Owners of a module can list all versions of TOs in modules that they own.

Typed object validation

Instances of TOs can be validated against type definitions registered with the WSS. Instances of TOs must pass this validation process to be stored in the WSS, thereby guaranteeing that WSS data is structurally valid.

Todo: Update this document to use the kb-sdk tools.

The WSS validates the TO instance against a JSON Schema V4 encoding of the TO definition. The JSON Schema encoding can be generated by the KBase Type Compiler (currently in branch `dev-prototypes`). In addition to matching the structure and type of data, additional constraints can be placed on TO validation through the use of Annotations (see *Typed object annotations*).

To generate JSON encodings of your TOs for review, checkout the `dev-prototypes` branch of the typecompiler and compile your typespec file with the `--jsonschema` option of the `compile_typespec` command. The JSON Schema encoding of each object definition is generated in the output location in a directory called jsonschema. The JSON Schema encoding is also available for all registered TO definitions via the WSS API or the `ws-typespec-list` command.

All TO instances pulled from the WSS are guaranteed to be valid instances of a TO definition as registered with the WSS. Therefore it is recommended that KBase services which require rigorous validation of complex data operate on data stored in the WSS (as opposed to passing the object by value and writing the validation code yourself). Note that full validation is not built into generated KBase client/server code, so it is not safe to assume that input data received directly from a type compiler generated client conforms to the specified type definitions in your API.

Additional technical details: The TO validation code is written in Java and is available in the workspace_deluxe KBase repo.

Typed object annotations

Annotations provide an infrastructure for attaching structured meta data to type definitions (and eventually to functions and modules). Such meta data is useful for specifying additional constraints on data types, interpreting data types within a particular context, and declaring structured information that can later be automatically indexed or searched, such as authorship of a function implementation.

Annotations are declared in the comment immediately preceding the definition of the TO. Thus, all annotations are always attached and viewable within the API documentation. Each annotation must be specified on its own line in the following format:

```
@[ANNOTATION] [INFO]
```

where `[ANNOTATION]` is the name of the annotation and `[INFO]` is any additional information, if any, required of the annotation. To provide a simple example which associates authorship information to a TO using the `@author` annotation:

```
/*
  Data type for my experimental data.
  @author John Scientist
*/
typedef structure {
    string name;
    list <int> results;
} MyExperimentData;
```

Currently supported type definition annotations

Optional annotation

Mark a specific field of a structure as an optional field. The optional annotation can only be declared where a structure is first defined. On validation of TO instances by the WSS, missing optional fields are permitted. If an optional field is present, however, the value of the field will be validated normally. Optional fields are defined as:

```
@optional [FIELD_NAME_1] [FIELD_NAME_2] ...
```

For example, the following annotation will declare that two fields of the structure are optional.:

```
/*
  @optional alias functional_assignments
*/
typedef structure {
    string name;
    string alias;
    string sequence;
    list <string> functional_assignments;
} Feature;
```

ID annotations

Mark a string as an ID that references another object or entity. ID annotations can only be associated to type definitions which resolve to a string. ID annotations are declared in the general form:

```
@id [ID_TYPE] [PARAMETERS]
```

where `[ID_TYPE]` specifies the type of ID and is required, and `[PARAMETERS]` provides additional information or constraints. `[PARAMETERS]` are always optional.

ID annotations are inherited when declaring a new `typedef` of a string that was already marked as an ID. If a new ID Annotation is declared in a `typedef`, it overrides any previous ID declaration.

Note that although `@id` annotations may be specified as any `ID_TYPE` and associated to any `typedef`, applications that consume type specifications (primarily the workspace at the time of writing) may only recognize specific `@id ID_TYPE` / `typedef` combinations.

The ID types currently supported are described below.

Workspace ID

```
@id ws [TYPEDEF_NAME] ...
```

The ID must reference a TO instance stored in the WSS. There are multiple valid ways to specify a workspace object, and all are acceptable. A reference path into the object graph may be provided by providing a string consisting of a list of references separated by semicolons.

Optionally, one or more type definition names can be specified indicating that the ID must point to a TO instance that is one of the specified types. The typedef with which the `@id` annotation is associated must be a string.

Example:

```
/*
   A reference to a genome.
   @id ws KB.MicrobialGenome KB.PlantGenome
*/
typedef string genome_id;
```

KBase ID

```
@id kb
```

The ID must reference a KBase ID which is typically registered in the ID service in a format such as "kb|type.XXX". No type checking on this field is performed, but the annotation may be used in the future so that users can automatically extract KBase IDs from typed objects.

Handle ID

```
@id handle
```

The ID must reference a Handle ID from the Handle Service. This is typically in the format KBH_XXX. When saving an object containing one or more handles to the WSS, the WSS checks that the handles are readable by the user before completing the save. Furthermore, the handle data is shared as the workspace object is shared. See *Shock integration with the workspace service* for more details.

External ID

```
@id external [SOURCE] ...
```

The ID must reference an entity in an external (i.e. outside of KBase) data store. The IDs are not verified or validated, but may be used in the future to allow users to automatically extract external IDs from typed objects. `[SOURCE]` provides an optional way to specify the external source. Currently there is no standard dictionary of sources.

Deprecated annotation

```
@deprecated [REPLACEMENT_TYPE]
```

The deprecated annotation is used to mark a type definition as deprecated, and provides a structured mechanism for indicating a replacement type if one exists. The deprecated annotation so far is only for documentation purposes, but may be used by the Workspace in the future to better display, list, or query workspace objects (e.g. list all objects of a type that is not deprecated).

Range annotation

```
@range [RANGE SPECIFICATION]
```

The range annotation is associated with either a float or int typedef and specifies the minimum and / or maximum value of the int or float. The `[RANGE SPECIFICATION]` is a tuple of the minimum and maximum numbers, separated by a comma. Omit the minimum or maximum to specify an infinite negative or positive range, respectively. Bracketing the `[RANGE SPECIFICATION]` with parentheses indicates the range extents are exclusive; square brackets or no brackets indicates an inclusive range.

Examples:

Range	Explanation
0, 30	Range from 0 - 30, inclusive
[0, 30]	Range from 0 - 30, inclusive
[0, 30)	Range from 0 - 30, including 0, excluding 30
(0,	Range from 0 - +inf, excluding 0
,30]	Range from -inf - 30, including 30

Example specification:

```
/*
   @range -4.5,7.6)
*/
typedef float my_float;

/*
   @range [2,10]
*/
typedef int my_int;
```

Metadata annotation

```
@metadata [CONTEXT] [ACTION] [as NAME]
```

The metadata annotation specifies data that an application should extract from a TO as metadata about the TO. Typically this metadata is very small compared to the TO and is therefore suitable for use when only a summary of the TO is necessary for an operation. As of this writing, the WSS uses the annotation to automatically generate user metadata for a TO.

The metadata annotation may only be associated with `structure` `typedef` s. Metadata annotations on nested `structure` s are ignored.

`[CONTEXT]` specifies where the metadata annotation is applicable. In the case of the WSS, the `[CONTEXT]` is `ws`. `[CONTEXT]` is always required.

`[ACTION]` specifies what metadata should be extracted and any operations to perform on said metadata. At minimum, the `[ACTION]` must provide the path (dot separated) to the item of interest. Note that the path may only proceed through `structure` `typedef` s, not `mapping` s or `list` s. A bare path must terminate at a primitive type - either a `string`, `int`, or `float`.

`[ACTION]` s may also specify a function to apply to the item specified by the path. Currently, the only available function is `length()`, which may be applied to `list` s, `mapping` s, `tuple` s, and `string` s. `length()` returns the number of items in a `list`, `mapping`, or `tuple`, or the length of a `string`.

`[as NAME]` allows specifying an optional `NAME` for the extracted metadata. If a `NAME` is not provided, the application will use the `[ACTION]` string as the metadata name. The `NAME` is entirety of the remainder of the line after "as".

Example:

```
/* Nested structure, metadata annotations have no effect here
   Cannot provide a path into the mapping in a metadata annotation
*/
typedef structure {
    mapping<string, string> strmap;
    int an_int;
} InnerStruct;

/*
   Specifies the metadata ("str" -> value of str in TO)
   @metadata ws str

   Specifies the metadata ("my rad string" -> value of str in TO)
   @metadata ws str as my rad string

   Specifies the metadata ("inner.an_int" -> value of inner.an_int in TO)
   @metadata ws inner.an_int

   Specifies the metadata ("length(str)" -> length of str in TO)
   @metadata ws length(str)

   Specifies the metadata ("num strings" -> # of items in inner.strmap)
   @metadata ws length(inner.strmap) as num strings

   Note that metadata paths cannot enter outerstrmap.
*/
typedef structure {
    InnerStruct inner;
    string str;
    mapping<string, string> outerstrmap;
} MyStruct;
```

2.2.4 Developing typed object definitions

Providing a comprehensive guide for developing type specifications (typespecs) for typed objects (TOs) in the Workspace Service (WSS) is far beyond the scope of this documentation, but provided here are some general guidelines and hints.

TO size and composition

- Generally speaking, the approach of translating each row from a traditional RDBMS into a single TO is very wrong. The major advantage of TOs is that they allow you to compose various related data into a single object.
- It is faster to save and load a single large TO as opposed to a many small TOs. Many small objects will also slow the WSS overall and increase the WSS index size.
  - The `get_objects2` method allows retrieving subsets of a TO from the WSS to provide the equivalent of retrieving a few small TOs rather than one large TO and then manually extracting the small TOs.
- TOs are currently limited to 1GB by the WSS.

- When contemplating TO design, consider how user interfaces might display workspaces and objects. Note that workspaces containing thousands of objects quickly become untenable.
- Objects which consist mostly of very long strings are usually much less useful when stored in the workspace than more structured data objects. Objects like this (for example DNA sequence or raw FASTA files) might be candidates for storage in Shock.

Very large objects

- Although in general, one larger object is better than many smaller objects, when objects are in the hundreds of megabytes they become less useful and more difficult to deal with.
  - One cannot realistically fetch a very large object (VLO) to a webpage.
- Even when using workspace functions to extract subdata from a VLO, the VLO must still be loaded from disk into the workspace service, which could take significant time.
- VLOs are slow to transfer in general.
- VLOs take a large amount of memory.
- VLOs can often take 3-20 times the size of the serialized object to represent in memory.
- Objects with large numbers of `mapping` s or `structure` s can use large amounts of resources due to repeated keys. Consider using `tuple` s instead of `mapping` s or `structure` s.

Annotations

TO to TO references (@id ws)

- TO to TO references using the `@id ws` annotation [see *ID annotations*] greatly enhance the utility of typed objects.
- For example, linking a data set TO to the genome TO that the data set references enforces and records the relationship in the workspace database.
- If a TO to be saved references a TO that doesn't exist, the error is caught prior to saving the TO in the workspace.
- If you have access to a TO, you can always access the TOs referenced by that TO, regardless of the workspace in which they're stored.
- However, there is a performance cost - each reference must be checked for existence in the database. For tens or even hundreds of references this cost is not high, but thousands or more unique references will likely slow saving of the TO.

@optional

- Avoid the `@optional` annotation whenever possible. In some cases its use is required, but every `@optional` annotation in a typespec makes the associated TOs more difficult to use for downstream programmers. If a typespec has no `@optional` annotations, a programmer knows exactly what data the TO contains and so the code to manipulate it can be simpler and therefore less buggy, easier to maintain, and less work to test.

2.2.5 Type checking error messages

This document provides explanations of several common type checking errors that may occur when attempting to save a typed object (TO) to the Workspace Service (WSS) that doesn't match the specified type.

Assume that the following spec has been released:

```
module AModule {

    /* @optional opt */
    typedef structure {
        list<mapping<string, int>> array_of_maps;
        int an_int;
        float a_float;
        string a_string;
        int opt;
    } AType;
};
```

The examples below show an example object, the error received, and an explanation of the error.

Missing property

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": 2}],
 "a_float": 1.4,
 "a_string": "s"
 }
```

WSS error: `object has missing required properties (["an_int"]), at /`

Explanation: The non-optional field an_int is missing.

Float vs. string

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": 2}],
 "an_int": "1",
 "a_float": "1",
 "a_string": "1"
 }
```

WSS error: `instance type (string) does not match any allowed primitive` `type (allowed: ["integer","number"]), at /a_float`

Explanation: The value for a_float must be a number, but was sent as a string.

Integer vs. string

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": 2}],
 "an_int": "1",
 "a_float": 1,
```

```
 "a_string": "1"
 }
```

WSS error: `instance type (string) does not match any allowed primitive` type (allowed: ["integer"]), at /an_int

Explanation: The value for an_int must be an integer, but was sent as a string.

Integer vs. float

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": 2}],
 "an_int": 1.4,
 "a_float": 1,
 "a_string": "1"
 }
```

WSS error: `instance type (number) does not match any allowed primitive` type (allowed: ["integer"]), at /an_int

Explanation: The value for an_int must be an integer, but was sent as a float.

String vs. integer

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": 2}],
 "an_int": 1,
 "a_float": 1.4,
 "a_string": 1
 }
```

WSS error: `instance type (integer) does not match any allowed primitive` type (allowed: ["string"]), at /a_string

Explanation: The value for a_string must be a string, but was sent as an integer.

Embedded

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": "2"}],
 "an_int": 1,
 "a_float": 1.4,
 "a_string": "s"
 }
```

WSS error: `instance type (string) does not match any allowed primitive` type (allowed: ["integer"]), at /array_of_maps/1/two

Explanation: The value of the two field in the subdocument in the second position of the array_of_maps array must be an integer, but was sent as a string.

Optional

JSON:

```
{"array_of_maps": [{"one": 1}, {"two": 2}],
 "an_int": 1,
 "a_float": 1.4,
 "a_string": "s",
 "opt": "1"
 }
```

WSS error: `instance type (string) does not match any allowed primitive` `type (allowed: ["integer"]), at /opt`

Explanation: The value of the optional field opt must be an integer, but was sent as a string. Note that in previous examples no error occurred even though the optional field was omitted.

2.2.6 Workspaces

Workspaces provide a means to collect multiple typed objects (TOs) into one container and share the container with other people. This documentation will demonstrate some of the most common operations on workspaces (see *Workspace API documentation* for the full API). It assumes that a functional client is available (see *Build and initialize the workspace client*). The examples use the Python client, but translating to other clients is trivial.

Creating workspaces

Create a workspace called MyWorkspace:

```
In [4]: ws.create_workspace({'workspace': 'MyWorkspace'})
Out[4]:
[12,                            # workspace numerical ID
 u'MyWorkspace',                # workspace name
 u'kbasetest',                  # workspace creator
 u'2015-12-13T20:48:00+0000',   # modification date of the workspace
 0,                             # number of objects created in this workspace
 u'a',                          # user's permission for the workspace
 u'n',                          # global permissions for the workspace
 u'unlocked',                   # whether the workspace is locked
 {}]                            # user provided metadata
```

Once created, a workspace's numerical ID is permanent and unchangeable. A locked workspace cannot be altered (other than making it world-readable).

Note that the object count is the total objects ever created in this workspace, not the currently existing objects.

Permissions

Permissions are coded according to the following table:

Permission	Allows
n	No access
r	Read access
w	Write access, see permissions of other users
a	Admin access, set permissions of other users

A workspace can have a description and arbitrary key-value metadata associated with it:

```
In [5]: ws.create_workspace({'workspace': 'MyOtherWorkspace',
   ...:                      'description': 'Workspace for other things',
   ...:                      'meta': {'contents': 'other things',
   ...:                               'project_id': '42'}
   ...:                      })
Out[5]:
[13,
 u'MyOtherWorkspace',
 u'kbasetest',
 u'2015-12-13T20:51:57+0000',
 0,
 u'a',
 u'n',
 u'unlocked',
 {u'contents': u'other things', u'project_id': u'42'}]
```

Retrieving information about workspaces

The workspace description and information list can be retrieved:

```
In [6]: ws.get_workspace_description({'id': 13}) # retrieving by ID
Out[6]: u'Workspace for other things'

In [11]: ws.get_workspace_info({'workspace': 'MyOtherWorkspace'})
Out[11]:
[13,
 u'MyOtherWorkspace',
 u'kbasetest',
 u'2015-12-13T20:51:57+0000',
 0,
 u'a',
 u'n',
 u'unlocked',
 {u'contents': u'other things', u'project_id': u'42'}]
```

Listing workspaces

Workspaces with at least read access can be listed:

```
In [8]: ws.list_workspace_info({})
Out[8]:
 [12,
  u'MyWorkspace',
  u'kbasetest',
  u'2015-12-13T20:48:00+0000',
  0,
  u'a',
  u'n',
  u'unlocked',
  {}],
 [13,
  u'MyOtherWorkspace',
  u'kbasetest',
  u'2015-12-13T20:51:57+0000',
```

```
  0,
  u'a',
  u'n',
  u'unlocked',
  {u'contents': u'other things', u'project_id': u'42'}]]
```

The list can be filtered in several ways. Here it's filtered by the user provided metadata:

```
In [10]: ws.list_workspace_info({'meta': {'project_id': '42'}})
Out[10]:
[[13,
  u'MyOtherWorkspace',
  u'kbasetest',
  u'2015-12-13T20:51:57+0000',
  0,
  u'a',
  u'n',
  u'unlocked',
  {u'contents': u'other things', u'project_id': u'42'}]
```

Sharing workspaces

Users with admin privileges with to a workspace can allow other users to read, write to, and administrate the workspace. These privileges apply to all objects contained in the workspace.

```
In [12]: ws.set_permissions({'workspace': 'MyWorkspace',
                             'users': ['kbasetest2'],
                             'new_permission': 'a'
                             })

In [13]: ws.set_permissions({'workspace': 'MyWorkspace',
                             'users': ['kbasetest8'],
                             'new_permission': 'r'
                             })

In [16]: ws.get_permissions_mass([{'id': 12},
                                  {'workspace': 'MyOtherWorkspace'}
                                  ])
Out[16]:
[{u'kbasetest': u'a', u'kbasetest2': u'a', u'kbasetest8': u'r'},
 {u'kbasetest': u'a'}]
```

2.2.7 Objects

This documentation describes some of the most common operations on objects in the workspace, with a focus on saving objects (see *Workspace API documentation* for the full API). It assumes that a functional client is available (see *Build and initialize the workspace client*). The examples use the Python client, but translating to other clients is trivial.

Warning: Objects saved to the WSS cannot contain binary data anywhere within their hierarchy. Binary data must be encoded (e.g. via Base64) prior to saving to the WSS.

Save an object

The object will be saved as the type `SimpleObjects.SimpleObject-1.0`, defined as:

```
In [4]: print ws.get_type_info('SimpleObjects.SimpleObject-1.0')['spec_def']
/*
@optional opt
*/
typedef structure {
  list<mapping<string, int>> array_of_maps;
  int an_int;
  float a_float;
  string a_string;
  int opt;
} SimpleObject;
```

Saving an object requires specifying either the name or id of the workspace, and for each object the name or id of the object, the type of the object, and the object data.

```
In [16]: obj = {'array_of_maps': [],
    ....:        'an_int': 42,
    ....:        'a_float': 6.02e-23,
    ....:        'a_string': 'towel'}

In [18]: ws.save_objects(
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'simple',
                           'type': u'SimpleObjects.SimpleObject-1.0',
                           'data': obj
                           }
                          ]
              })
Out[18]:
[[1,                                     # object numerical ID
  u'simple',                             # object name
  u'SimpleObjects.SimpleObject-1.0',     # object type
  u'2015-12-14T04:11:55+0000',           # save or copy date of the object
  1,                                     # version of the object
  u'kbasetest',                          # user that saved or copied the object
  12,                                    # numerical workspace ID
  u'MyWorkspace',                        # workspace name
  u'6b76d883ffa1357e52e1020594317dd7',   # MD5 digest of the object
  70,                                    # size of the object in bytes
  {}]]                                   # user provided metadata
```

Once created, an object's numerical ID is permanent and unchangeable.

The MD5 is calculated *after* any references are translated (see below) and the object `structure` and `mapping` keys are sorted.

Saving an object that does not match the typespec causes an error:

```
In [23]: obj['a_string'] = 42

In [24]: ws.save_objects(
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'simple2',
                           'type': u'SimpleObjects.SimpleObject-1.0',
                           'data': obj
```

```
                            }
                           ]
              })
---------------------------------------------------------------------------
ServerError                               Traceback (most recent call last)
<ipython-input-11-91a2d5e7f85e> in <module>()
      3               'objects': [{'name': 'simple2',
      4                            'type': u'SimpleObjects.SimpleObject-1.0',
----> 5                            'data': obj
      6                            }
      7                           ]

*snip*

ServerError: JSONRPCError: -32500. Object #1, simple2 failed type checking:
instance type (integer) does not match any allowed primitive type (allowed: ["string
↪"]), at /a_string
*snip*
```

Saving an object with `null` s (or in Python's case `None` s) where an `int`, `float`, or `string` is expected is allowed:

```
In [21]: obj = {'array_of_maps': [],
                'an_int': None,
                'a_float': None,
                'a_string': None}

In [22]: ws.save_objects(
             {'id': 12,
              'objects': [{'name': 'nullobj',
                           'type': u'SimpleObjects.SimpleObject-1.0',
                           'data': obj
                           }
                          ]
              })
Out[22]:
[[3,
  u'nullobj',
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T22:58:55+0000',
  1,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'0eb7130429570c6fe23017091df0a654',
  65,
  {}]]
```

Save a new version

Providing an existing name or ID when saving an object causes the creation of a new object version:

```
In [20]: obj = {'array_of_maps': [],
                'an_int': 42,
                'a_float': 6.02e-23,
                'a_string': 'hoopty frood'}
```

```
In [22]: ws.save_objects(
             {'id': 12,
              'objects': [{'objid': 1,
                           'type': u'SimpleObjects.SimpleObject-1.0',
                           'data': obj
                           }
                          ]
              })
Out[22]:
[[1,                                      # same object ID
  u'simple',                              # same name
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T04:22:38+0000',
  2,                                      # new version
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'8aba51168748e7a7a91847f510ce2807',    # new MD5
  77,                                     # 7 more bytes wasted
  {}]]
```

Save an object with metadata

As with workspaces, arbitrary key-value metadata can be associated with objects:

```
In [27]: ws.save_objects(
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'simple3',
                           'type': u'SimpleObjects.SimpleObject-1.0',
                           'data': obj,
                           'meta': {'Eccentrica': 'Gallumbits',
                                    'Wowbagger': 'Prolonged'
                                    }
                           }
                          ]
              })
Out[27]:
[[2,
  u'simple3',
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T04:43:21+0000',
  1,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'8aba51168748e7a7a91847f510ce2807',
  77,
  {u'Eccentrica': u'Gallumbits', u'Wowbagger': u'Prolonged'}]]
```

Save an object with provenance

Provenance data may be saved along with the object data as a list of provenance actions (PAs). Each PA represents a step taken to convert a data unit into another - for example, passing a genome sequence to a server which returns a metabolic model for that sequence. The PA contains fields for recording how an object was generated. See the *Workspace API documentation* for the full specification, but some common fields are:

Field	Description
time	The time the action took place
service	The name of the service that produced the object
service_ver	The version of the service
method	The method called on the service
description	A free text description of the action

Some fields require special explanation. The `intermediate_incoming` and `intermediate_outgoing` fields allow linking the outputs of one PA with the inputs of the next. The list of PAs is assumed to be in the order the actions took place, so, for example, if workspace object `A` was passed to a service method as `X.process(A)` which produced the object tuple `[B, C]`, and those results were passed to a service method as `Y.dothing(C, B)` which produced the object `D`, the provenance list might look like:

```
pl = [{'service': 'X',
       'method': 'process',
       'intermediate_outgoing': ['B', 'C']
       },
      {'service': 'Y',
       'method': 'dothing',
       'intermediate_incoming': ['C', 'B']
       'method_params': ['C', 'B']
       }
      ]
```

`B` and `C`, in this example, are merely symbols that describe the ordering of the inputs and outputs of each step and any permutations of those orders from step to step. Any unique names could be used.

The `input_ws_objects` field allows specifying workspace objects that were used in the creation of the current object and therefore are part of its provenance. In the example above, object `A` is part of the provenance of object `D`, and should therefore be specified in `input_ws_objects`:

```
pl = [{'service': 'X',
       'method': 'process',
       'intermediate_outgoing': ['B', 'C'],
       'input_ws_objects': ['MyWorkspace/2/2']
       },
      {'service': 'Y',
      ...
```

In this case, `A` was the 2nd version of object ID `2` in `MyWorkspace`. The name or ID of the workspace and object may be used in the reference string. Names will always be translated to IDs by the WSS before the provenance is saved, since IDs are permanent and names are not.

For example:

```
In [27]: ps = [{'description': 'assemble paired end reads',
                'input_ws_objects': ['MyWorkspace/simple/1'],
                'method': 'annotatePairedReads',
                'method_params': [{'objname': 'simple',
                                   'workspace': 'MyWorkspace',
                                   'ver': 1
                                   }
                                  ],
                'service': 'Annotation',
                'service_ver': '2.1.3',
                'time': '2015-12-15T22:58:55+0000'
                }
```

```
               ]

In [30]: ws.save_objects(
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'simpleWithProv',
                           'type': u'SimpleObjects.SimpleObject-1.0',
                           'data': obj,
                           'provenance': ps
                           }
                          ]
              })
Out[30]:
[[4,
  u'simpleWithProv',
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T23:44:35+0000',
  2,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'6b76d883ffa1357e52e1020594317dd7',
  70,
  {}]]
```

If the object is retrieved, it can be seen that the `resolved_ws_objects` field has been added to the provenance. This field contains the translated object references supplied in `input_ws_objects`:

```
In [32]: ws.get_objects2({'objects':
             [{'ref': 'MyWorkspace/simpleWithProv'}]})['data']
Out[32]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-14T23:44:35+0000',
  u'creator': u'kbasetest',
  u'data': {u'a_float': 6.02e-23,
   u'a_string': u'towel',
   u'an_int': 42,
   u'array_of_maps': []},
  u'extracted_ids': {},
  u'info': [4,
   u'simpleWithProv',
   u'SimpleObjects.SimpleObject-1.0',
   u'2015-12-14T23:44:35+0000',
   2,
   u'kbasetest',
   12,
   u'MyWorkspace',
   u'6b76d883ffa1357e52e1020594317dd7',
   70,
   {}],
  u'provenance': [{u'description': u'assemble paired end reads',
    u'external_data': [],
    u'input_ws_objects': [u'MyWorkspace/simple/1'],
    u'method': u'annotatePairedReads',
    u'method_params': [{u'objname': u'simple',
                        u'workspace': u'MyWorkspace'
                        u'ver': 1}],
    u'resolved_ws_objects': [u'12/1/1'],
    u'service': u'Annotation',
```

```
    u'service_ver': u'2.1.3',
    u'time': u'2015-12-15T22:58:55+0000'}],
  u'refs': []}]
```

Saving provenance with objects is optional, but strongly encouraged.

Warning: The WSS does not inherently know anything about the provenance of the objects it stores, and cannot evaluate the reliability or completeness of the provenance. It is entirely up to the user or application storing the objects to ensure accurate and complete provenance. Clearly the provenance in the examples above is fradulent.

Save an object with dependency references

The following types will be used to demonstrate saving objects with dependency references:

```
In [52]: print ws.get_module_info({'mod': 'SimpleObjects'})['spec']
module SimpleObjects {

    /* @optional opt */
    typedef structure {
        list<mapping<string, int>> array_of_maps;
        int an_int;
        float a_float;
        string a_string;
        int opt;
    } SimpleObject;

    typedef structure {
        int i;
        string thing;
    } SimplerObject;

    /* @id ws */
    typedef string ref;

    /* @id ws SimpleObjects.SimplerObject */
    typedef string typedref;

    typedef structure {
        ref r;
        string thing;
    } RefObject;

    typedef structure {
        typedref r;
        string thing;
    } TypeRefObject;
};
```

Saving an object with a dependency reference required by the typespec is just like saving any other object:

```
In [57]: refobj = {'r': 'MyWorkspace/simple',
                   'thing': 'this object has a reference'
                   }

In [58]: ws.save_objects(
```

```
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'ref',
                           'type': u'SimpleObjects.RefObject-2.0',
                           'data': refobj,
                           }
                          ]
              })
Out[58]:
[[6,
  u'ref',
  u'SimpleObjects.RefObject-2.0',
  u'2015-12-15T03:12:41+0000',
  1,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'44e0ef9dff44c4840ddf77abbfc555bd',
  52,
  {}]]

In [59]: ws.get_objects2({'objects':
             [{'workspace': 'MyWorkspace', 'name': 'ref'}]})['data']
Out[59]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-15T03:12:41+0000',
  u'creator': u'kbasetest',
  u'data': {u'r': u'12/1/2',
            u'thing': u'this object has a reference'
            },
  u'extracted_ids': {},
  u'info': [6,
   u'ref',
   u'SimpleObjects.RefObject-2.0',
   u'2015-12-15T03:12:41+0000',
   1,
   u'kbasetest',
   12,
   u'MyWorkspace',
   u'44e0ef9dff44c4840ddf77abbfc555bd',
   52,
   {}],
  u'provenance': [],
  u'refs': [u'12/1/2']}]
```

Note that the reference in the saved object was translated to a permanent reference, and that the references are extracted into the `refs list` in the returned data.

If the referenced object is not accessible to the user saving the object, the save will fail. If the save succeeds, the referent will be forever accessible to users with access to the referencing object as described previously.

Types may specify that a reference must point to an object with a specific type, as in the `TypeRefObject` type. In this case, saving with a reference that does not point to an object with type `SimpleObjects.SimplerObject` will fail:

```
In [73]: ws.save_objects(
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'typedref',
                           'type': u'SimpleObjects.TypeRefObject',
```

```
                            'data': refobj,
                            }
                           ]
              })
---------------------------------------------------------------------------
ServerError                                Traceback (most recent call last)
<ipython-input-73-80b8ab6aabd0> in <module>()
      3               'objects': [{'name': 'typedref',
      4                            'type': u'SimpleObjects.TypeRefObject',
----> 5                            'data': refobj,
      6                            }
      7                           ]

*snip*

ServerError: JSONRPCError: -32500. Object #1, typedref has invalid
reference: The type SimpleObjects.SimpleObject-1.0 of reference
MyWorkspace/simple in this object is not allowed - allowed types are
[SimpleObjects.SimplerObject] at /r
```

List objects

Listing objects is similar to listing workspaces:

```
In [74]: ws.list_objects({'ids': [12]})
Out[74]:
[[2,
  u'simple3',
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T22:58:01+0000',
  1,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'8aba51168748e7a7a91847f510ce2807',
  77,
  None],

 *snip*

 [4,
  u'simpleWithProv',
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T23:44:35+0000',
  2,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'6b76d883ffa1357e52e1020594317dd7',
  70,
  None]]
```

Listing also works by type:

```
In [78]: ws.list_objects({'type': 'SimpleObjects.RefObject'})
Out[78]:
```

```
[[6,
  u'ref',
  u'SimpleObjects.RefObject-2.0',
  u'2015-12-15T03:12:41+0000',
  1,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'44e0ef9dff44c4840ddf77abbfc555bd',
  52,
  None]]
```

A large number of filters exist for `list_objects` - see the *Workspace API documentation* for comprehensive coverage. In this example, the list is filtered by the object metadata:

```
In [82]: ws.list_objects({'ids': [12],
                          'meta': {'Wowbagger': 'Prolonged'},
                          'includeMetadata': 1
                          })
Out[82]:
[[2,
  u'simple3',
  u'SimpleObjects.SimpleObject-1.0',
  u'2015-12-14T22:58:01+0000',
  1,
  u'kbasetest',
  12,
  u'MyWorkspace',
  u'8aba51168748e7a7a91847f510ce2807',
  77,
  {u'Eccentrica': u'Gallumbits', u'Wowbagger': u'Prolonged'}]]
```

2.2.8 Subsetting objects

When retrieving objects from the WSS, the user may specify which parts of the object to retrieve. This is useful for quickly retrieving small portions of large objects (to a webpage, for example) rather than having to fetch the entire object which might be hundreds of megabytes.

Note that performing subsetting on small objects may provide little to no benefit, and in some cases may be slower, since the WSS has to parse the serialized object rather than directly returning the serialized form to the client.

As usual, it is assumed that a functional client is available (see *Build and initialize the workspace client*). The examples use the Python client, but translating to other clients is trivial. Only the most common cases are covered - see the *Workspace API documentation* for complete coverage.

For the examples, the following spec was used:

```
In [16]: print ws.get_type_info("SubSetExample.SubSetExample")['spec_def']
typedef structure {
    mapping<string, mapping<string, string>> map;
    list<mapping<string, string>> array;
} SubSetExample;
```

The object in question:

```
In [20]: data = {'map': {'mid1': {'id': 'id1', 'stuff': 'foo'},
   ....:                 'mid2': {'id': 'id2', 'stuff': 'bar'}
```

```
   ....:                 },
   ....:         'array': [{'id': 'id1', 'stuff': 'foo'},
   ....:                   {'id': 'id2', 'stuff': 'bar'},
   ....:                   {'id': 'id3', 'stuff': 'baz'}
   ....:                   ]
   ....:         }

In [24]: ws.save_objects(
             {'workspace': 'MyWorkspace',
              'objects': [{'name': 'subsetexample',
                           'type': u'SubSetExample.SubSetExample',
                           'data': data,
                           }
                          ]
              })
Out[24]:
[[1,
  u'subsetexample',
  u'SubSetExample.SubSetExample-1.0',
  u'2015-12-16T03:57:03+0000',
  1,
  u'kbasetest',
  13,
  u'MyWorkspace',
  u'f9449880abc5722c7add56e773544719',
  168,
  {}]]
```

Get the contents of a single key of the mapping:

```
In [11]: ws.get_objects2({'objects':
             [{'workspace': 'MyWorkspace',
               'name': 'subsetexample',
               'included': ['/map/mid1']
               }
              ]})['data']
Out[25]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-16T03:57:03+0000',
  u'creator': u'kbasetest',
  u'data': {u'map': {u'mid1': {u'id': u'id1', u'stuff': u'foo'}}},
  u'extracted_ids': {},
  u'info': [1,
   u'subsetexample',
   u'SubSetExample.SubSetExample-1.0',
   u'2015-12-16T03:57:03+0000',
   1,
   u'kbasetest',
   13,
   u'MyWorkspace',
   u'f9449880abc5722c7add56e773544719',
   168,
   {}],
  u'provenance': [],
  u'refs': []}]
```

Get all the `stuff` fields from the mapping:

```
In [39]: ws.get_objects2({'objects':
             [{'workspace': 'MyWorkspace',
               'name': 'subsetexample',
               'included': ['/map/*/stuff']
               }
              ]})['data']
Out[39]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-16T04:04:59+0000',
  u'creator': u'kbasetest',
  u'data': {u'map': {u'mid1': {u'stuff': u'foo'},
    u'mid2': {u'stuff': u'bar'}}},
  u'extracted_ids': {},
  u'info': [1,
   u'subsetexample',
   u'SubSetExample.SubSetExample-1.0',
   u'2015-12-16T04:04:59+0000',
   2,
   u'kbasetest',
   13,
   u'MyWorkspace',
   u'24cd918528461efcb9d6f6a02c3a7965',
   168,
   {}],
  u'provenance': [],
  u'refs': []}]
```

Get all the `id` fields from the array:

```
In [33]: ws.get_objects2({'objects':
             [{'workspace': 'MyWorkspace',
               'name': 'subsetexample',
               'included': ['/array/[*]/id']
               }
              ]})['data']
Out[33]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-16T04:04:59+0000',
  u'creator': u'kbasetest',
  u'data': {u'array': [{u'id': u'id1'}, {u'id': u'id2'}, {u'id': u'id3'}]},
  u'extracted_ids': {},
  u'info': [1,
   u'subsetexample',
   u'SubSetExample.SubSetExample-1.0',
   u'2015-12-16T04:04:59+0000',
   2,
   u'kbasetest',
   13,
   u'MyWorkspace',
   u'24cd918528461efcb9d6f6a02c3a7965',
   168,
   {}],
  u'provenance': [],
  u'refs': []}]
```

Get the first and third elements of the array (note that the returned array is compressed to only 2 cells, but the ordering of the source array is maintained):

```
In [35]: ws.get_objects2({'objects':
             [{'workspace': 'MyWorkspace',
               'name': 'subsetexample',
               'included': ['/array/2', '/array/0']
               }
              ]})['data']
Out[35]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-16T04:04:59+0000',
  u'creator': u'kbasetest',
  u'data': {u'array': [{u'id': u'id1', u'stuff': u'foo'},
    {u'id': u'id3', u'stuff': u'baz'}]},
  u'extracted_ids': {},
  u'info': [1,
   u'subsetexample',
   u'SubSetExample.SubSetExample-1.0',
   u'2015-12-16T04:04:59+0000',
   2,
   u'kbasetest',
   13,
   u'MyWorkspace',
   u'24cd918528461efcb9d6f6a02c3a7965',
   168,
   {}],
  u'provenance': [],
  u'refs': []}]
```

The previous two calls can be used to find and fetch portions of an array. First fetch the parts of the subdocuments to be used to determine which portions of the array are desired, and next fetch the array subdocuments of interest based on processing the first query. This approach may or may not be faster than fetching the entire array, so the user should test their particular use case.

2.2.9 Traversing object references

Object to object references, whether dependency or provenance references, not only indicate, respectively, objects that are required to compute on or understand the origin of the referencing object, but also provide permanent access to the referenced objects. The philosophy behind this design is that data is useless if it is incomplete (dependency references) or has unknown origins (provenance references). The object to object references form a graph structure which, in the context of the workspace, allows unlimited traversal *in the direction of the references*. Methods exist to provide traversal capabilities in the opposite direction, but only to objects to which the user has direct access.

As usual, it is assumed that a functional client is available (see *Build and initialize the workspace client*). The examples use the Python client, but translating to other clients is trivial. Only the most common cases are covered - see the *Workspace API documentation* for complete coverage.

The following examples use this new type specification:

```
In [12]: print user1client.get_module_info({'mod': 'Ref'})['spec']
module Ref {

    /* @id ws */
    typedef string aref;

    typedef structure {
        aref ref;
    } RefType;
};
```

See *Save an object* for the `SimpleObjects` specification and *Save an object with dependency references* for details regarding saving objects with references.

In the interest of simplicity, saving the example objects is not shown. User 1 (kbasetest2) saved two objects in their workspace, one of which contains a reference to the other:

```
In [28]: user1client.get_objects2({'objects':
                                   [{'ref': 'user1ws/simple'},
                                   {'ref': 'user1ws/refobj1'}
                                   ]})['data']
Out[28]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-18T04:13:15+0000',
  u'creator': u'kbasetest2',
  u'data': {u'a_float': 6.02e-23,
   u'a_string': u'towel',
   u'an_int': 42,
   u'array_of_maps': []},
  u'extracted_ids': {},
  u'info': [1,
   u'simple',
   u'SimpleObjects.SimpleObject-1.0',
   u'2015-12-18T04:13:15+0000',
   1,
   u'kbasetest2',
   13,
   u'user1ws',
   u'6b76d883ffa1357e52e1020594317dd7',
   70,
   {}],
  u'provenance': [],
  u'refs': []},
 {u'copy_source_inaccessible': 0,
  u'created': u'2015-12-18T04:14:33+0000',
  u'creator': u'kbasetest2',
  u'data': {u'ref': u'13/1/1'},   # points at object above
  u'extracted_ids': {},
  u'info': [2,
   u'refobj1',
   u'Ref.RefType-1.0',
   u'2015-12-18T04:14:33+0000',
   1,
   u'kbasetest2',
   13,
   u'user1ws',
   u'160cf883f216b170f5d2074652e1bf5d',
   16,
   {}],
  u'provenance': [],
  u'refs': [u'13/1/1']}]
```

This workspace is readable to User 2 (kbasetest8):

```
In [30]: user1client.get_permissions_mass(
             {'workspaces': [{'workspace': 'user1ws'}]})
Out[30]: [{u'kbasetest2': u'a', u'kbasetest8': u'r'}]
```

As such, User 2 saved an object that references User 1's `refobj1`:

```
In [31]: user2client.get_objects2(
             {'objects': [{'ref': 'user2ws/refobj2'}]})['data']
Out[31]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-18T04:16:20+0000',
  u'creator': u'kbasetest8',
  u'data': {u'ref': u'13/2/1'},
  u'extracted_ids': {},
  u'info': [1,
   u'refobj2',
   u'Ref.RefType-1.0',
   u'2015-12-18T04:16:20+0000',
   1,
   u'kbasetest8',
   14,
   u'user2ws',
   u'ad38c241c9a46bb940fb4574a343b3c5',
   16,
   {}],
  u'provenance': [],
  u'refs': [u'13/2/1']}]
```

If User 1 now sets `user1ws` to unreadable, and worse, deletes the second object:

```
In [32]: user1client.set_permissions({'workspace': 'user1ws',
                                      'users': ['kbasetest8'],
                                      'new_permission': 'n'})
In [34]: user1client.delete_objects([{'ref': 'user1ws/refobj1'}])
```

... as expected User 2 now cannot access the object referenced by their `refobj2` object, which renders it useless.

```
In [35]: user2client.get_objects2(
             {'objects': [{'ref': 'user1ws/refobj1'}]})['data']
---------------------------------------------------------------------------
ServerError                               Traceback (most recent call last)
<ipython-input-35-7c5faa02c112> in <module>()
----> 1 user2client.get_objects([{'ref': 'user1ws/refobj1'}])

*snip*

ServerError: JSONRPCError: -32500. Object refobj1 cannot be accessed: User
kbasetest8 may not read workspace user1ws
```

However, using the `get_objects2` method and providing the path from an accessible object to the desired object, User 2 can still retrieve the hidden/deleted objects, and thus use `refobj2`. The path can be deduced from the references in each object:

```
In [51]: user2client.get_objects2(
             {'objects': [{'ref': 'user2ws/refobj2'}]})['data']
Out[51]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-18T04:16:20+0000',
  u'creator': u'kbasetest8',
  u'data': {u'ref': u'13/2/1'},
  u'extracted_ids': {},
  u'info': [1,
```

```
   u'refobj2',
   u'Ref.RefType-1.0',
   u'2015-12-18T04:16:20+0000',
   1,
   u'kbasetest8',
   14,
   u'user2ws',
   u'ad38c241c9a46bb940fb4574a343b3c5',
   16,
   {}],
  u'provenance': [],
  u'refs': [u'13/2/1']}]

In [52]: user2client.get_objects2(
             {'objects': [{'ref': 'user2ws/refobj2',
                           'obj_path': [{'ref': '13/2/1'}]
                           }
                          ]})
Out[52]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-18T04:14:33+0000',
  u'creator': u'kbasetest2',
  u'data': {u'ref': u'13/1/1'},
  u'extracted_ids': {},
  u'info': [2,
   u'refobj1',
   u'Ref.RefType-1.0',
   u'2015-12-18T04:14:33+0000',
   1,
   u'kbasetest2',
   13,
   u'user1ws',
   u'160cf883f216b170f5d2074652e1bf5d',
   16,
   {}],
  u'provenance': [],
  u'refs': [u'13/1/1']}]

In [53]: user2client.get_objects2(
             {'objects': [{'ref': 'user2ws/refobj2',
                           'obj_path': [{'ref': '13/2/1'},
                                        {'ref': '13/1/1'}
                                        ]
                          ]})
Out[53]:
[{u'copy_source_inaccessible': 0,
  u'created': u'2015-12-18T04:13:15+0000',
  u'creator': u'kbasetest2',
  u'data': {u'a_float': 6.02e-23,
   u'a_string': u'towel',
   u'an_int': 42,
   u'array_of_maps': []},
  u'extracted_ids': {},
  u'info': [1,
   u'simple',
   u'SimpleObjects.SimpleObject-1.0',
   u'2015-12-18T04:13:15+0000',
   1,
```

```
   u'kbasetest2',
   13,
   u'user1ws',
   u'6b76d883ffa1357e52e1020594317dd7',
   70,
   {}],
  u'provenance': [],
  u'refs': []}]
```

It is also possible for User 1 to find objects that reference their objects if they are readable and not in the deleted state:

```
In [54]: user1client.undelete_objects([{'ref': '13/2'}])

In [55]: user1client.list_referencing_objects([{'ref': 'user1ws/simple'}])
Out[55]:
[[[2,
   u'refobj1',
   u'Ref.RefType-1.0',
   u'2015-12-18T04:14:33+0000',
   1,
   u'kbasetest2',
   13,
   u'user1ws',
   u'160cf883f216b170f5d2074652e1bf5d',
   16,
   {}]]]
```

Attempting to list User 2's object, which references `refobj1` and is unreadable by User 1, is not possible:

```
In [56]: user1client.list_referencing_objects([{'ref': 'user1ws/refobj1'}])
Out[56]: [[]]
```

Note that although not shown, provenance references work exactly the same way. This example is, of course, very simple - a single object could have many references, and those objects may also have many references, et cetera.

2.2.10 Locking and publishing workspaces

A workspace administrator may lock a workspace, preventing (most) further changes. If a locked workspace is globally readable, it is considered published - a user might wish to publish a workspace that contains supplemental information for a publication, for example, so that the publication editors can see that the information is permanently recorded.

> **Warning:** Once a workspace is locked, it can never be unlocked, even by a server administrator.

As usual, it is assumed that a functional client is available (see *Build and initialize the workspace client*). The examples use the Python client, but translating to other clients is trivial.

Lock a workspace (see *Creating workspaces* and *Save an object* for information on creating workspaces and saving objects):

```
In [8]: ws.lock_workspace({'workspace': 'MyWorkspace'})
Out[8]:
[12,
 u'MyWorkspace',
 u'kbasetest',
 u'2015-12-20T01:09:49+0000',
```

```
 2,
 u'a',
 u'n',
 u'locked',
 {}]
```

The following methods are not allowed on locked workspaces:

- `alter_workspace_metadata`
- `delete_objects`
- `undelete_objects`
- `delete_workspace`
- `hide_objects`
- `unhide_objects`
- `lock_workspace`
- `rename_object`
- `rename_workspace`
- `revert_object`
- `save_object` (note this function is deprecated)
- `save_objects`
- `set_workspace_description`

`set_permissions` does work.

Additionally `set_global_permission` may only be used to make the workspace globally readable. A locked, globally readable workspace may not be made private:

```
In [9]: ws.set_global_permission({'workspace': 'MyWorkspace',
                                  'new_permission': 'r'})

In [10]: ws.set_global_permission({'workspace': 'MyWorkspace',
                                   'new_permission': 'n'})
---------------------------------------------------------------------------
ServerError                               Traceback (most recent call last)
<ipython-input-10-c700ea19406a> in <module>()
----> 1 ws.set_global_permission({'workspace': 'MyWorkspace',
                                  'new_permission': 'n'})

*snip*

ServerError: JSONRPCError: -32500. The workspace with id 12, name MyWorkspace,
 is locked and may not be modified
```

2.2.11 Shock integration with the workspace service

Overview

Shock is a data storage system originally designed for metagenomics data. As such, it is designed for fast reads and writes of data structured, for the most part, as linear arrays of strings (such as FASTA biologic sequence files), but

more generally bytestreams.

In contrast, the WSS is designed for storing the hierarchical data objects used in most programming languages and specifically as specified by the KIDL. In many cases, bytestream and sequential data may be more efficiently stored in and retrieved from Shock, and in the cases of data > 1GB, cannot be stored in the WSS (see *Workspace limits*).

This document describes how to use the Handle Service to link WSS objects to Shock nodes, such that when the object is shared via the Workspace API, the linked Shock nodes are shared (more specifically, made readable) as well. If a data type developer merely stored a Shock node ID in a workspace object as a string, sharing the object would not share the underlying Shock node, and sharees would not be able to access the Shock data.

Warning: Handles, by their nature, are not necessarily permanent. The owner of the data referenced by a handle could remove or otherwise make it inaccessible at any time.

Warning: Shock nodes shared by the workspace are not unshared if the workspace object containing the Shock node handle is unshared. The Shock nodes can always be unshared via the Shock API.

Warning: Sharing workspace objects containing handles to Shock nodes shares the nodes as well. If a workspace object is copied into a user's workspace and that workspace is made public, the Shock nodes are set to publically readable.

Resources

Workspace typed objects describes how to create workspace types.

Shock API

Handles

You can create handles to data in Shock via the Handle Service. The Handle Service manages pointers, or handles, to arbitrary pieces of data, and provides unique IDs for each handle you create. The type definition for a handle is:

```
typedef string HandleId;
typedef structure {
    HandleId hid;
    string file_name;
    string id;
    string type;
    string url;
    string remote_md5;
   string remote_sha1;
} Handle;
```

For Shock handles, the fields are defined to be:

Member	Description
hid	a unique id assigned to a Handle by the Handle service
file_name	the file's name
id	the shock node id
type	the string "shock"
url	the shock server url
remote_md5	the md5 of the shock data
remote_sha1	unused

The remainder of the document covers the procedure for linking a Workspace object to a Shock node.

Step 1 - save data to Shock with a Handle in the Handle Service

Method 1 - use the Perl HandleService client

The Perl HandleService client makes creating a handle to shock data simple - it uploads the file to Shock and creates a handle in one step:

```
$hs = Bio::KBase::HandleService->new();
$handle = $hs->upload($filename);
```

The ID of the handle can then be retrieved via the hid field:

```
$hid = $handle->{hid};
```

If the Shock data already exists, merely persist a handle you create (leave the hid field empty for this usage):

```
$hid = $hs->persist_handle($handle);
```

Method 2 - pre-existing Shock data without the HandleService client

Save data to Shock

Here it is assumed that you are familiar with the Shock API, but as an example:

```
$ curl -X POST -H "Authorization: OAuth $TOKEN" -F upload=@important_data.txt https://
↪[shock url]/node

{"status":200,"data":{"id":"e9f1b8b2-0012-47a9-89ef-fb8fad5a2a5e",
 "version":"e757db0fff0398841505c314179e85f8","file":{"name":
 *snip*
 "2014-08-01T13:12:47.091885252-07:00","type":"basic"},"error":null}
```

Create one or more handles to Shock data

If you're working in a language other than Perl, you can use the AbstractHandle client to persist handles. Here's a python example:

```
In [1]: from biokbase.AbstractHandle.Client import AbstractHandle
In [2]: ah = AbstractHandle('https://[handle url]', user_id='kbasetest',␣
↪password=[redacted])

In [3]: handle = {'type': 'shock', 'url':
                  'https://[shock url]',
                  'id': 'e9f1b8b2-0012-47a9-89ef-fb8fad5a2a5e'
                  }

In [4]: ah.persist_handle(handle)
Out[4]: u'KBH_8'
```

Method 3 - new Shock data without the HandleService client

Create one or more handles for your data

Use the Handle Service new_handle method to create handles:

```
In [48]: from biokbase.AbstractHandle.Client import AbstractHandle
In [49]: ah = AbstractHandle('https://[handle url]',
                             user_id='kbasetest', password=[redacted])

In [50]: ah.new_handle()
Out[50]:
{u'file_name': None,
 u'hid': u'KBH_12',
 u'id': u'70ff43ff-db14-405a-bc03-e4dc46860833',
 u'type': u'shock',
 u'url': u'https://[shock url]'}
```

Save data to the Shock node referenced by the handle

Again, using the Shock API:

```
$ curl -X PUT -H "Authorization: OAuth $KBASETEST_TOKEN" -F upload=@important_data.
↪txt https://[shock url]/node/70ff43ff-db14-405a-bc03-e4dc46860833

{"status":200,"data":{"id":"70ff43ff-db14-405a-bc03-e4dc46860833",
 "version":"458bf368a56ffeeb0a33faa2349b0b7e","file":{"name":
 *snip*
 "2014-08-02T10:32:04.278684787-07:00","type":"basic"},"error":null}
```

Step 2 - create a Workspace type for your data

If a type specification doesn't already exist for your data, you will need to create one. The key point is that you must make the Workspace Service aware that your data contains one or more Handle IDs. This is done via the `@id handle` annotation (see *ID annotations*):

```
/* @id handle */
typedef string HandleId;

/* @optional file_name
```

```
   @optional remote_sha1
   @optional remote_md5
*/
typedef structure {
    HandleId hid;
    string file_name;
    string id;
    string type;
    string url;
    string remote_md5;
    string remote_sha1;
} Handle;
```

Depending on your requirements, you may wish to mark some of the fields optional as above. All the Workspace service absolutely requires is the handle ID (`hid`), although marking the `url` or `id` as optional is unwise, as the `Handle` will not contain enough information for users to retrieve the shock data.

We then can embed Handles in our data type:

```
/* @optional handles */
typedef structure {
    Handle handle;
    list<Handle> handles;
    string veryimportantstring;
    int veryimportantint;
} VeryImportantData;
```

At this point type creation proceeds along normal lines (see *Workspace typed objects*).

Step 3 - save data with embedded Handles to the Workspace

Saving data with embedded handles is identical to saving any other WSS object. This example assumes the the type described in the previous section is present in the VeryImportantModule module and has been registered and released.

```
In [1]: from biokbase.workspace.client import Workspace
In [3]: ws = Workspace('https://[workspace url]',
                       user_id='kbasetest', password=[redacted])

In [13]: handle1 = {'hid': 'KBH_8',
                    'id': 'e9f1b8b2-0012-47a9-89ef-fb8fad5a2a5e',
                    'url': 'https://[shock url]',
                    'type': 'shock'
                    }
In [14]: handle2 = {'hid': 'KBH_5',
                    'id': 'ed732169-31a6-4acb-a59c-401d95cc7e3e',
                    'url': 'https://[shock url]',
                    'type': 'shock'
                    }
In [20]: vip_data = {'handle': handle1,
                     'handles': [handle2],
                     'veryimportantstring': 'My word, I am important',
                     'veryimportantint': 42
                     }

In [23]: ws.save_objects(
             {'workspace': 'foo',
              'objects': [{'name': 'foo',
```

```
                            'type': 'VeryImportantModule.VeryImportantData-2.0',
                            'data': vip_data
                            }
                           ]
              })
Out[23]:
[[1,
  u'foo',
  u'VeryImportantModule.VeryImportantData-2.0',
  u'2014-08-01T20:20:58+0000',
  13,
  u'kbasetest',
  2,
  u'foo',
  u'e62152ed3bd328e3001083d0d230ecc0',
  302,
  {}]]
```

During the save, the Workspace checks with the Handle Service to confirm the user owns the Shock data. If such is not the case, the save will fail.

Step 4 - share data in the Workspace

Sharing data works completely normally.

Step 5 - retrieve the data from the Workspace

Retrieving the data from the workspace also works normally, but there's a couple of important points. When calling the `get_objects`, `get_objects2`, `get_referenced_objects`, `get_object_subset`, or `get_object_provenance` methods:

- The Handle IDs found in the object are returned in the output as strings, and
- The Workspace makes a request to the Handle Service such that the caller of the method is given read access to the data referenced by the handles embedded in the object.

This means that, mostly invisibly, the shock nodes embedded via Handles in a Workspace object are shared as the object is shared.

```
In [18]: ws.get_objects2({'objects': [{'ref': 'foo/foo'}]})['data']
Out[18]:
[{u'created': u'2014-08-01T20:20:58+0000',
  u'creator': u'kbasetest',
  u'data': {u'handle': {u'hid': u'KBH_8',
                        u'id': u'e9f1b8b2-0012-47a9-89ef-fb8fad5a2a5e',
                        u'type': u'shock',
                        u'url': [shock_url]
                        },
            u'handles': [{u'hid': u'KBH_5',
                          u'id': u'ed732169-31a6-4acb-a59c-401d95cc7e3e',
                          u'type': u'shock',
                          u'url': [shock_url]
                          }
                         ],
            u'veryimportantint': 42,
            u'veryimportantstring': u'My word, I am important'
```

```
            },
  u'extracted_ids': {u'handle': [u'KBH_8',
                                 u'KBH_5'
                                 ]
                     },
  u'info': [1,
            u'foo',
            u'VeryImportantModule.VeryImportantData-2.0',
            u'2014-08-01T20:20:58+0000',
            13,
            u'kbasetest',
            2,
            u'foo',
            u'e62152ed3bd328e3001083d0d230ecc0',
            302,
            {}],
  u'provenance': [],
  u'refs': []}]
```

The Shock data can then be retrieved via the Shock API using the handle information embedded in the object.

If a node has been deleted, the handle service is uncontactable, or some other error occurs, the workspace will still return the workspace object. However, the error will be embedded in the returned data structure. The handle_error field will contain a brief description of the error, and the handle_stacktrace field will contain the full stacktrace. If these fields are populated the ACLs of some or all of the Shock nodes embedded in the object could not be updated.

```
In [26]: ws.get_objects2({'objects': [{'ref': 'foo/foo'}]})['data']
Out[26]:
[{u'created': u'2014-08-08T00:07:10+0000',
  u'creator': u'kbasetest',
  u'data': {u'handles': [u'KBH_5', u'KBH_6']},
  u'extracted_ids': {u'handle': [u'KBH_6', u'KBH_5']},
  u'handle_error': u'The Handle Manager reported a problem while attempting to set␣
↪Handle ACLs: Unable to set acl(s) on handles KBH_6, KBH_5',
  u'handle_stacktrace': u'us.kbase.common.service.ServerException: Unable to set␣
↪acl(s) on handles KBH_6, KBH_5\n
  \tat us.kbase.common.service.JsonClientCaller.jsonrpcCall(JsonClientCaller.
↪java:269)\n

  *snip*

  \tat java.lang.Thread.run(Thread.java:724)\n',
  u'info': [1,
            u'foo',
            u'ListHandleIds.HandleList-0.1',
            u'2014-08-08T00:07:12+0000',
            5,
            u'kbasetest',
            334,
            u'foo',
            u'd98067db987ccdf5321819b39f73440d',
            29,
            {}
            ],
  u'provenance': [],
  u'refs': []
  }
 ]
```

2.2.12 Workspace limits

This document provides a list of limits of the WSS.

Limits

Parameter	Limit
Maximum RPC call size	1.005GB
Maximum object size	1GB
Maximum total size of returned objects	1GB
Maximum provenance size	1MB
Maximum user metadata size	16000B
Maximum total size of user metadata key / value pair	900B
Maximum total size of autometadata key / value pair	900B
Maximum memory use for sorting objects	200MB
Maximum object_infos returned by list_objects	10000
Maximum workspace references per save	100000

Notes on sorting

The workspace service sorts the contents of all objects before MD5 calculations, serialization, and storage.

When sorting objects, object mapping and structure keys in a single path from the object root to a single object leaf are stored in memory at one time. The memory limit applies to these keys plus the memory required for the object itself.

Objects > 100MB in size are dumped to disk, so the maximum memory allowed for keys is 200MB. Objects < 100MB are kept in memory, so the maximum memory allowed is 200MB - object size.

Thus, objects may violate this limit if 1) they have very large maps, 2) have many very large keys in the same map, or 3) have very deeply nested maps (which probably still need to be fairly large).

As a point of reference, sorting a 550MB Network object required only ~10MB of memory for keys.

Notes on workspace references

The workspace service supports a maximum of 100,000 object references (e.g. a reference specified by @id ws in a typespec) per saveObjects() call. The references may be in a single object, or spread across many objects.

References that are duplicated in a single object only count once towards this limit.

2.2.13 FAQ

I can release a mapping / list / string / etc. type, but I can't save anything as that type. Why not?

The workspace is only intended to store typed objects - e.g. a class or structure in the context of a programming language. This confusion arises because releasing a type has two conflated meanings. Firstly, releasing a type means that types in other KIDL specs can import and use the type. Secondly, releasing a type means that the workspace can save objects as that type - but the workspace only supports saving KIDL structures.

Note: In the future we may separate these meanings into two separate operations, one for releasing a type for use in other types, and one for releasing structures for saving to the workspace.

Why do I keep getting type checking errors in Perl when I know I created the right type?

Perl is weakly typed - e.g. types are coerced as necessary depending on the context. This means that you may put an integer into a data structure, but it could be coerced to a string in a subsequent operation. The data is currently transported to the Workspace Service in JSON, which is strongly typed, as is the Workspace Service type checker, so if type coercion occurs the type checker will see (for example) a string when an integer is expected and the data will fail type checking.

It may help to dump the data object to JSON and inspect it just before sending it to the workspace and look, for example, for quoted integers (in other words, strings) where the KIDL type specification requires integers.

Note that `Data::Dumper` coerces all data to strings before dumping and is therefore not useful for debugging type checking problems.

2.2.14 Known user-facing bugs

Todo: Find or create some user-facing bugs

2.2.15 Workspace API documentation

- KIDL spec as HTML
- KIDL spec as text
- Perl
- Java

Todo: Add js and py documentation when the typecompiler / kb-sdk generates it

2.3 Server Administrator Documentation

2.3.1 Build, configure, and deploy

These instructions assume the reader is familiar with the process of deploying a KBase module, including the runtime and dev_container, and has access to a system with the KBase runtime installed. These instructions are based on the `kbase-image-v26` runtime image.

Unlike many modules the WSS can be built and tested outside the `dev_container`, but the `dev_container` is required to build and test the scripts. These instructions are for deploying the server and so do not address the scripts. Building outside the `dev_container` means the Makefile uses several default values for deployment - if you wish to use other values deploy from the `dev_container` as usual.

Build the workspace service

First checkout the `dev_container`:

```
/kb$ sudo git clone https://github.com/kbase/dev_container
Cloning into 'dev_container'...
remote: Counting objects: 1097, done.
remote: Total 1097 (delta 0), reused 0 (delta 0), pack-reused 1097
Receiving objects: 100% (1097/1097), 138.81 KiB, done.
Resolving deltas: 100% (661/661), done.
```

Note: In the v26 image, `/kb` is owned by `root`. As an alternative to repetitive `sudo` s, `chown /kb` to the user.

Bootstrap and source the user environment file, which sets up Java and Perl paths which the WSS build needs:

```
/kb$ cd dev_container/
/kb/dev_container$ sudo ./bootstrap /kb/runtime/
/kb/dev_container$ source user-env.sh
```

Now the WSS may be built. If building inside the `dev_container` all the dependencies from the `DEPENDENCIES` file are required, but to build outside the `dev_container`, only the `jars` and `workspace_deluxe` repos are necessary:

```
~$ mkdir kb
~$ cd kb
~/kb$ git clone https://github.com/kbase/workspace_deluxe
Cloning into 'workspace_deluxe'...
remote: Counting objects: 21961, done.
remote: Compressing objects: 100% (40/40), done.
remote: Total 21961 (delta 20), reused 0 (delta 0), pack-reused 21921
Receiving objects: 100% (21961/21961), 21.42 MiB | 16.27 MiB/s, done.
Resolving deltas: 100% (13979/13979), done.

~/kb$ git clone https://github.com/kbase/jars
Cloning into 'jars'...
remote: Counting objects: 1466, done.
remote: Total 1466 (delta 0), reused 0 (delta 0), pack-reused 1466
Receiving objects: 100% (1466/1466), 59.43 MiB | 21.49 MiB/s, done.
Resolving deltas: 100% (626/626), done.

~/kb$ cd workspace_deluxe/
~/kb/workspace_deluxe$ make
*snip*
```

`make` will build:

- A workspace client jar in `/dist/client`
- A workspace server jar in `/dist`
- This documentation in `/docs`

Note: If the build fails due to a sphinx error, sphinx may require an upgrade to >= 1.3:

```
$ sudo pip install sphinx --upgrade
```

Service dependencies

The WSS requires MongoDB 2.4+ to run. The WSS may optionally use:

- Shock as a file storage backend.
- The Handle Service b9de699 + and Handle Manager 3e60998 + to allow linking workspace objects to Shock nodes (see *Shock integration with the workspace service*).

The WSS has been tested against the auth2 branch of the KBase fork of Shock version 0.9.6 (e9f0e1618e265042bf5cb96429995b5e6ec0a06a), and against MongoDB versions 2.4.14, 2.6.11, 3.0.8, and 3.2.1. 3.0+ versions were tested with and without the WiredTiger storage engine.

Please see the respective service documentation to set up and run the services required.

Note: The alternative to Shock as a file storage backend is MongoDB GridFS. GridFS is simpler to set up, but locks the entire database when writing files. Since the workspace can consume very large files, this can cause a significant impact on other database operations.

Configuration

There are two sources of configuration data for the WSS. The first is contained in the `deploy.cfg` file in the repository root (see *Configuration parameters*). Copy the provided `deploy.cfg.example` file to `deploy.cfg` to create the file. These parameters may change from invocation to invocation of the workspace service. The second is contained in the workspace MongoDB database itself and is set once by the configuration script (see *Configuration script*).

Warning: `deploy.cfg` contains several sets of credentials, and thus should be protected like any other file containing unencryted passwords or tokens. It is especially important to protect the password / token that the WSS uses to talk to Shock (`backend-secret` or `backend-token`) as if access to that account is lost, the new account owner has access to all the workspace object data, and recovery will be extremely time consuming (use shock admin account to change all the acls for every WSS owned object to the new account). At minimum, only the user that runs the WSS (which should **not** be `root`) should have read access to `deploy.cfg`. Also be aware that the `deploy.cfg` contents are copied to, by default, `/kb/deployment/deployment.cfg` when the workspace is deployed from the `dev_container`.

Configuration parameters

mongodb-host

Required: Yes

Description: Host and port of the MongoDB server, eg. localhost:27017

mongodb-database

Required: Yes

Description: Name of the workspace MongoDB database

mongodb-user

Required: If the MongoDB instance requires authorization

Description: Username for an account with readWrite access to the MongoDB database

mongodb-pwd

Required: If the MongoDB instance requires authorization

Description: Password for an account with readWrite access to the MongoDB database

auth-service-url

Required: Yes

Description: URL of the KBase authentication service

globus-url

Required: Yes

Description: URL of the Globus Nexus v1 authentication API

ignore-handle-service

Required: If not using handles

Description: Set to anything (`true` is good) to not use handles. In this case attempting to save an object with a handle will fail. Delete or leave blank to use handles (the default).

handle-service-url

Required: If using handles

Description: The URL of the Handle Service

handle-manager-url

Required: If using handles

Description: The URL of the Handle Manager

handle-manager-token

Required: If using handles

Description: Credentials for the account approved for Handle Manager use

ws-admin

Required: No

Description: the user name for a workspace administrator. This name, unlike names added via the `administer` API call, is not permanently stored in the database and thus the administrator will change if this name is changed and the server restarted. This administrator cannot be removed by the `administer` API call.

backend-token

Required: If using Shock as the file backend

Description: Token for the file backend user account used by the WSS to communicate with the backend. The user name is stored in the database after being determined by the configuration script.

port

Required: Yes

Description: The port on which the service will listen

server-threads

Required: Yes

Description: See *server-threads*

min-memory

Required: Yes

Description: See *min-memory and max-memory*

max-memory

Required: Yes

Description: See *min-memory and max-memory*

temp-dir

Required: Yes

Description: See *temp-dir*

mongodb-retry

Required: No

Description: Startup MongoDB reconnect retry count. The workspace will try to reconnect 1/s until this limit has been reached. This is useful for starting the Workspace automatically after a server restart, as MongoDB can take quite a while to get from start to accepting connections. The default is no retries.

dont-trust-x-ip-headers

Required: No

Description: When `true`, the server ignores the `X-Forwarded-For` and `X-Real-IP` headers. Otherwise (the default behavior), the logged IP address for a request, in order of precedence, is 1) the first address in `X-Forwarded-For`, 2) `X-Real-IP`, and 3) the address of the client.

Configuration script

Before starting the WSS for the first time, the database must be configured with information about the type database and file backend. This information travels with the MongoDB database because it is intrinsic to the overall data store - once a type database and file backend are chosen, they cannot be changed later without causing massive data inconsistency.

Prior to configuring the database, MongoDB must be running. If using Shock as a backend, Shock must be running.

To configure the database, run the initialization script, which will step the user through the process:

```
~/kb/workspace_deluxe$ cd administration/
~/kb/workspace_deluxe/administration$ ./initialize.py
Current configuration file:
mongodb-host=localhost
mongodb-database=workspace
handle-service-url=
handle-manager-url=
handle-manager-token=
auth-service-url=https://kbase.us/services/auth/api/legacy/KBase/Sessions/Login/
globus-url=https://kbase.us/services/auth/api/legacy/KBase
ws-admin=workspaceadmin
backend-token=
port=7058
server-threads=20
min-memory=10000
max-memory=15000
temp-dir=ws_temp_dir
mongodb-retry=0

Keep this configuration? [y - keep]/n - discard: n
Discarding current local configuration.
Please enter value for mongodb-host: localhost
Please enter value for mongodb-database: ws_db
Does mongodb require authentication? [y - yes]/n - no: n
Ok, commenting out authorization information.
Attempting to connect to mongodb database "ws_db" at localhost... Connected.
Please enter the name of the mongodb type database: ws_db_types
Choose a backend:  [s - shock]/g - gridFS: s
Please enter the url of the shock server: http://localhost:7044
```

```
Please enter an authentication token for the workspace shock user account: [redacted]
Validating token with auth server at https://kbase.us/services/auth/api/legacy/KBase/
↪Sessions/Login/
Successfully set DB configuration:
type_db=ws_db_types
backend=shock
shock_location=http://localhost:7044/
shock_user=gaprice

Saving local configuration file:
mongodb-host=localhost
mongodb-database=ws_db
handle-service-url=
handle-manager-url=
handle-manager-token=
auth-service-url=https://kbase.us/services/auth/api/legacy/KBase/Sessions/Login/
globus-url=https://kbase.us/services/auth/api/legacy/KBase
ws-admin=workspaceadmin
backend-token=[redacted]
port=7058
server-threads=20
min-memory=10000
max-memory=15000
temp-dir=ws_temp_dir
mongodb-retry=0

Configuration saved.
```

Note that the configuration script will only alter the `mongodb-*` and `backend-secret` parameters. Other parameters must be altered through manually editing `deploy.cfg`.

Also, do not, under any circumstances, use `kbasetest` as the account with which the WSS will communicate with Shock.

Once the database is started and `deploy.cfg` is filled in to the user's satisfaction, the server may be deployed and started.

Deploy and start the server

To avoid various issues when deploying, `chown` the deployment directory to the user. Alternatively, chown `/kb/` to the user, or deploy as root.

```
~/kb/workspace_deluxe$ sudo mkdir /kb/deployment
~/kb/workspace_deluxe$ sudo chown ubuntu /kb/deployment
~/kb/workspace_deluxe$ make deploy
*snip*
Makefile:53: Warning! Running outside the dev_container - scripts will not be␣
↪deployed or tested.
```

Since the service was deployed outside of the `dev_container`, the service needs to be told where `deploy.cfg` is located. When built in the `dev_container`, the contents of `deploy.cfg` are automatically copied to a global configuration and this step is not necessary.

```
~/kb/workspace_deluxe$ export KB_DEPLOYMENT_CONFIG=~/kb/workspace_deluxe/deploy.cfg
```

Next, start the service. If using Shock or the Handle services, ensure they are up and running before starting the WSS.

```
~/kb/workspace_deluxe$ /kb/deployment/services/workspace/start_service
Creating domain Workspace at /kb/deployment/services/workspace/glassfish_domain
Using default port 4848 for Admin.
Using default port 8080 for HTTP Instance.
*snip*
No domain initializers found, bypassing customization step
Domain Workspace created.
Domain Workspace admin port is 4848.
Domain Workspace allows admin login as user "admin" with no password.
Command create-domain executed successfully.
Starting domain Workspace
Waiting for Workspace to start .......
Successfully started the domain : Workspace
domain  Location: /kb/deployment/services/workspace/glassfish_domain/Workspace
Log File: /kb/deployment/services/workspace/glassfish_domain/Workspace/logs/server.log
Admin Port: 4848
Command start-domain executed successfully.
Removing options []
Setting option -Xms10000m
Removing options ['-Xmx512m']
Setting option -Xmx15000m
Restarting Workspace, please wait
Successfully restarted the domain
Command restart-domain executed successfully.
Creating property KB_DEPLOYMENT_CONFIG=/home/ubuntu/kb/workspace_deluxe/deploy.cfg
Command create-system-properties executed successfully.
Command create-virtual-server executed successfully.
Command create-threadpool executed successfully.
Command create-http-listener executed successfully.
server.network-config.network-listeners.network-listener.http-listener-7058.thread-
↪pool=thread-pool-7058
Command set executed successfully.
server.network-config.protocols.protocol.http-listener-7058.http.timeout-seconds=1800
Command set executed successfully.
Application deployed with name app-7058.
Command deploy executed successfully.
The server started successfully.
```

Stop the service:

```
~/kb/workspace_deluxe$ /kb/deployment/services/workspace/stop_service
Domain Workspace exists at /kb/deployment/services/workspace/glassfish_domain,␣
↪skipping creation
Domain Workspace is already running on port 4848
Command undeploy executed successfully.
Command delete-http-listener executed successfully.
Command delete-threadpool executed successfully.
Command delete-virtual-server executed successfully
```

Note that the `stop_service` script leaves the Glassfish server running. `kill` the Glassfish instance to completely shut down the server.

If any problems occur, check the glassfish logs (by default at `/kb/deployment/services/workspace/glassfish_domain/Workspace/logs/server.log` and system logs (on Ubuntu, at `/var/log/syslog`). If the JVM can't start at all (for instance, if the JVM can't allocate enough memory), the glassfish logs are the most likely place to look. If the JVM starts but the workspace application does not, the system logs should provide answers.

2.3.2 Test

Prior to running workspace tests, the workspace source files must be built. See *Build, configure, and deploy*.

In order to run tests:

- MongoDB must be installed, but not necessarily running.
- Shock must be installed, but not necessarily running.
- MySQL must be installed, but not necessarily running.
  - AppArmor must be configured to allow spawning of a mysql instance with files in non-default locations by the user running tests.
- The Handle Service must be installed, but not necessarily running.
- The Handle Manager must be installed, but not necessarily running.

See *Service dependencies* for more information about these test dependencies.

Next, copy the `test.cfg.example` file to `test.cfg` and fill in appropriately.

Finally:

```
make test
```

The tests currently take 20-30 minutes to run.

2.3.3 Workspace resource requirements

Several configuration variables define resources which are assigned to the WSS.

Configuration variables

Variable	Default	Notes
temp-dir	ws_temp_dir	Change to an appropriate location.
server-threads	20	
min-memory	10000	In MB.
max-memory	15000	In MB.

temp-dir

temp-dir determines where the workspace writes temporary files. The workspace is by default configured to need no more than 80GB of space at one time (see *Disk usage* below). The faster the drive on which the temp files directory is located, the faster the workspace will process large TOs.

server-threads

server-threads determines how many threads the server will run, which determines the maximum number of concurrent serviced connections. If more than this number of connections occur at the same time, they will be processed in the order received. **server-threads** dictates how much memory and disk space is needed for the server as a whole - see *Memory usage* and *Disk usage* below.

min-memory and max-memory

min-memory and **max-memory** set the minimum and maximum memory the Glassfish server, as a whole, will use (e.g. they're JVM parameters). It is assumed no other services run on the Glassfish server.

Memory usage

The workspace currently uses up to 400MB per call for saving data:

Amount	Use	If exceeded
100MB	Storage of the raw rpc data as bytes.	RPC call is dumped to disk.
100MB	Storage of sorted, relabeled TOs	All TO data is dumped to disk.
200MB	Memory for sorting & intermediate data per TO (processed serially). See *Notes on sorting*.	Intermediate data is dumped to disk or an error is returned if sorting takes too much memory.

Returning data is simpler - 300MB is allocated for all TO data, and any TO data exceeding this limit is dumped to disk.

Provenance and user provided metadata are not included in these limits but are expected to be small.

Thus, to be safe the minimum memory for the server should be set to 500MB per thread (thus the default 10GB for a 20 thread server).

Note: In the future, we hope to add a thread queue that detects free memory so that more threads can run when the memory load is not high (which is expected to be the case most of the time).

Disk usage

Disk usage is currently configured to use up to 3GB per call for saving data.

Amount	Use	If exceeded
1GB	Storage of the raw rpc data as bytes.	The server throws an error.
1GB	Storage of sorted, relabeled TOs	The server throws an error.
1GB	Storage of intermediate sort files	The server throws an error.

Returning data is configured to use no more than 2GB. Thus, to be perfectly safe, 4GB per server thread of temporary disk space should be allocated (thus 80GB for a 20 thread server).

2.3.4 Administration interface

This document describes the administration functions available via the `administer` API call. All administration calls, including running standard workspace operations like `create_workspace`, go through `administer` to avoid accidental use of administrative powers when calling the API or using scripts (similar to `sudo`).

First initialize a workspace client with administrator credentials:

```
from biokbase.workspace.client import Workspace
wsadmin = Workspace('https://kbase.us/services/ws', user_id=[user], password=[pwd])
```

Note: These examples use the Python client, but translating the commands to other languages is trivial.

Managing administrators

Adding an administrator:

```
wsadmin.administer({'command': 'addAdmin', 'user': 'lolcats'})
```

Listing administrators:

```
wsadmin.administer({'command': 'listAdmins'})
[u'lolcats', u'superadminman']
```

Removing administrators:

```
wsadmin.administer({'command': 'removeAdmin', 'user': 'lolcats'})
```

Note: The administrator specified in the `deploy.cfg` file cannot be removed by this method. See *Configuration parameters*.

Managing module ownership requests

See *Typed object registration & versioning*.

List module ownership requests:

```
wsadmin.administer({'command': 'listModRequests'})
[{u'moduleName': u'KBaseLolCats',
  u'ownerUserId': u'jkbaumohl',
  u'withChangeOwnersPrivilege': True}]
```

Accept module ownership request:

```
wsadmin.administer({'command': 'approveModRequest', 'module': 'KBaseLolCats'})
```

Reject module ownership request:

```
wsadmin.administer({'command': 'denyModRequest', 'module': 'KBaseLolCats'})
```

Managing workspaces

Change the owner of a workspace:

The `setWorkspaceOwner` command is more complex than the commands seen so far. It takes a map with a `param` key that maps to a map with the keys:

- `wsi` - a `WorkspaceIdentity` as specified in the API specification. Required.
- `new_user` - the user who will own the workspace. Required.
- `new_name` - the new name of the workspace. Optional.

Example:

```
wsadmin.administer(
    {'command': 'setWorkspaceOwner',
     'params': {'wsi': {'workspace': 'someuser:lolcats'},
                'new_user': 'jkbaumohl'
                }
     })
[3303,
 u'jkbaumohl:lolcats',
 u'jkbaumohl',
 u'2015-12-13T00:45:06+0000',
 0,
 u'a',
 u'n',
 u'unlocked',
 {}]
```

Note that the workspace is automatically renamed such that the user prefix matches the new user.

Note: Only a workspace administrator can change workspace ownership.

List all workspace owners:

```
wsadmin.administer({'command': 'listWorkspaceOwners'})
[u'auser',
 u'anotheruser',
 u'yetanotheruser',
 u'jkbaumohl']
```

General workspace commands

The `administer` interface allows running normal WSS API methods while acting as a different user (except in a few cases, see below). The commands all have the same basic structure:

```
wsadmin.administer(
    {'command': [method name inCamelCase],
     'params':  [parameters of the method per the API specification]
     'user':    [username under which the command will run]
    })
```

The methods currently available are:

Method	`user` required
createWorkspace	yes
setPermissions	no
getPermissions (DEPRECATED)	optional (1)
getPermissionsMass	no
getWorkspaceInfo	no
getObjectInfo	no (2)
getObjectHistory	no
getObjects	no (3)
setGlobalPermission	yes
saveObjects	yes
listWorkspaces	yes
listWorkspaceIDs	yes
listObjects	optional (4)
deleteWorkspace	no
undeleteWorkspace	no
grantModuleOwnership	no
removeModuleOwnership	no

1. If omitted, returns the permissions as if the user is an administrator of the workspace.
2. Parameters are as get_object_info3.
3. Parameters are as get_objects2.
4. If omitted, returns all objects requested, but at least one and no more than 1000 workspaces must be specified.

Example usage:

```
wsadmin.administer(
    {'command': 'createWorkspace',
     'params': {'workspace': 'morelolcats',
                'description': 'Golly, I really love lolcats.'
                },
     'user': 'jkbaumohl'
     })
[3304,
 u'morelolcats',
 u'jkbaumohl',
 u'2015-12-13T01:16:50+0000',
 0,
 u'a',
 u'n',
 u'unlocked',
 {}]

wsadmin.administer(
    {'command': 'getPermissions',
     'params': {'id': 3304},
     'user': 'superadminman'
     })
{u'superadminman': u'n'}

wsadmin.administer(
    {'command': 'setPermissions',
     'params': {'id': 3304,
                'new_permission': 'w',
```

```
                'users': ['superadminman']
                }
     })

wsadmin.administer(
    {'command': 'getPermissions',
     'params': {'id': 3304},
     'user': 'superadminman'})
{u'jkbaumohl': u'a', 'superadminman': u'w'}
```

2.3.5 Known administration bugs

- The WSS occasionally fails to start after a redeploy without restarting Glassfish, usually after 25-30 redeploys. Workaround by killing and restarting Glassfish.
- Due to application level locks in the type database portion of the server, only one instance of the server can be run at once.

Note: In the future the type service may be separated from the workspace service, which would mean the workspace service could run multiple instances. The vast majority of the load is on the workspace service.

2.4 Developer documentation

2.4.1 Contributions and branches

All pull requests should go to the `dev-candidate` branch or a feature branch.

Branches:

- `dev-candidate` - work in progress goes here, not stable, tests may not pass.
- `develop` - All tests pass. `dev-candidate` is merged here when features are ready for release. Ready for integration testing.
- `staging` - as dev.
- `master` - All tests pass, code is production ready.

`develop` deploys to `ci.kbase.us` while `staging` deploys to `next.kbase.us`. Generally speaking, most development would occur on `develop`, but because most of `ci` would break if the workspace breaks, `develop` must be kept stable.

2.4.2 Recompiling the generated code

To compile, simply run `make compile`. The kb-sdk executable must be in the system path.

2.4.3 Deploying the Worspace Service locally

These instructions are known to work on Ubuntu 16.04 LTS.

1. Install pymongo v2.8, mongodb v3.4.* and GlassFish v3.1.*.

```
$ sudo pip install pymongo==2.8
```

The GlassFish download URL is http://www.oracle.com/technetwork/middleware/glassfish/downloads/ogs-3-1-1-downloads-439803.html. The unix shell script (ogs-3.1.2.2-unix.sh) install of GlassFish is simple since the application configuration is also handled during install. Follow the wizard instructions to complete the GlassFish installation. Leave all config values to default values. The rest of this playbook assumes that you have the mongo and glassfish binaries of these applications set in your environment path variable.

2. Start mongodb. Open a new terminal and create a directory for your mongo data (if one does not already exist)

```
$ mkdir ~/mongodata
$ mongod --dbpath ~/mongodata
```

3. Set up the workspace for deployment in another terminal.

Note: If you are using Oracle Java 8, the javadoc command may throw errors and warnings. Add the following linter argument line to the javadoc command in build.xml to suppress these warnings and errors.

```
<javadoc access="protected" author="false" classpathref="compile.classpath"
  destdir="${doc}" nodeprecated="false" nodeprecatedlist="false"
  noindex="false" nonavbar="false" notree="false"
  source="1.7" splitindex="true" use="true" version="true">
  <arg line="-Xdoclint:none"/>    <!-- ADD THIS LINE -->
  <link href="http://download.oracle.com/javase/8/docs/api/"/>
  ....
</target>
```

Then run make.

```
$ make
```

Set up a fake kbase directory with a softlink to glassfish within it.

```
$ cd ../
$ mkdir fakekb
$ cd fakekb
$ ln -s ~/glassfish3
$ gedit glassfish3/glassfish/config/osgi.properties
```

Add this fix at the end of the osgi.properties file -

```
# fix for java 8
jre-1.8=${jre-1.7}
```

Get latest version of dev-candidate branch from git.

```
$ cd ../workspace_deluxe
$ git checkout dev-candidate
$ git pull
```

Configure the service for deployment. The instructions here assume the deployment is tied to the CI environment.

```
$ cp deploy.cfg.example deploy.cfg
$ gedit deploy.cfg
```

Make the following changes -

```
auth-service-url = https://ci.kbase.us/services/auth/api/legacy/KBase/Sessions/Login
globus-url = https://ci.kbase.us/services/auth/api/legacy/globus/
ws-admin = [YOUR_NAME]
# Note: ignore-handle-service does not exist and needs to be added
ignore-handle-service = true
```

4. Initialize and start the workspace service.

```
$ export KB_DEPLOYMENT_CONFIG=[ABSOLUTE_PATH_TO_deploy.cfg]
$ make deploy TARGET=[ABSOLUTE_PATH_TO_fakekb_DIR] DEPLOY_RUNTIME=[ABSOLUTE_PATH_TO_
↪fakekb_DIR]
$ cd administration
$ python ./initialize.py
Keep this configuration? yes
Does mongodb require authentication? no
Please enter the name of your mongodb type database: ws_types
Choose a backend: g
$ cd ..
$ [PATH_TO_FAKE_KB]/services/workspace/start_service
```

Note: If workspace service does not start successfully, tail /var/log/syslog for errors.

5. Check if the workspace service is working properly by creating a workspace service client, verifying workspace service version and creating a new workspace.

```
$ cd [PATH_TO_YOUR_WORKSPACE_DIR]/lib
$ ipython

In [1]: from biokbase.workspace.client import Workspace
In [2]: my_ci_token = 'YOUR CI TOKEN'
In [4]: ws = Workspace("http://localhost:7058", token=my_ci_token)
In [5]: ws.ver()
Out[5]: u'0.8.0-dev4'
In [6]: ws.create_workspace({'workspace': 'myws'})
Out[7]:
[1,
u'myws',
...
]
```

2.4.4 Release checklist

- Update the version in `docsource/conf.py`
- Update the version in the generated server java file
- Update release notes
- Update documentation if necessary.
- Ensure tests cover changes. Add new tests if necessary.
- Run tests against supported versions of MongoDB and Shock.
- Tag the release in git with the new version
- Merge `dev-candidate` to `develop`

- When satisfied with CI testing (work with devops here), merge `develop` to `staging`
- When satisfied with testing on `next.kbase.us` merge `staging` to `master`.

2.5 Workspace service release notes

2.5.1 VERSION: 0.8.0 (Released TBD)

BACKWARDS INCOMPATIBILITIES:

- The `undelete_workspace` method has been removed. Workspaces are now considered to be permanently deleted.
- Building and running the service now requires Java 8.
- The `getPermissions` administration command, like the `get_permissions` method, is now deprecated.

ADMIN NOTES:

- **Two new indexes have been added to the workspace versions mongo collection:**
  - the index `{savedby: 1}` with no options
  - the index `{ws: 1, id: 1, ver: -1}` with `{unique: 1}`
- The workspaces collection name index has been made sparse. The index must be changed before deploying this version.
- Added docker file & code for pushing docker image in a travis build.

NEW FEATURES:

- Adds a workspace event listener API. Event listeners must implement the `us.kbase.workspace.listener.WorkspaceEventListenerFactory` and `WorkspaceEventListener` interfaces. Specify listeners to be loaded on start up in the `deploy.cfg` file (see `deploy.cfg.example` for an example). See `us.kbase.workspace.test.listener.NullListenerFactory` for an example implementation.
- Added the `list_workspace_ids` method.
- Added the `listWorkspaceIDs` administration command.
- Added the `getPermissionsMass` administration command.
- Added the `getWorkspaceInfo` administration command.
- Added the `listObjects` administration command.
- Added the `getObjectInfo` administration command.
- Added the `getObjectHistory` administration command.
- Added the `getObjects` administration command.
- `list_objects` will now sort the output if no filters other than the object id filters are applied. The sort order is workspace id ascending, object id ascending, and version descending.

UPDATED FEATURES / MAJOR BUG FIXES:

- A user name is now optional for the `getPermissions` administration command.
- Fixed a bug where the administrator `setWorkspaceOwner` command in very specific cases could allow setting an illegal workspace name.

- Fixed a bug where an admin could delete a locked workspace.
- Removed `kbase-admin` credentials from the `deploy.cfg` file as they're obsolete after the conversion to auth2.
- The credentials for the Handle Manager service in the `deploy.cfg` file now require a token.
- The credentials for the file backend in the `deploy.cfg` file now require a token.
- Fixed a bug where performing a permissions search for a readable, deleted object with an incoming reference from a readable, non-deleted object would fail with a deleted object exception.
- Fixed a bug that could cause workspace clones to fail under certain conditions.

2.5.2 VERSION: 0.7.1 (Released 6/22/17)

UPDATED FEATURES / MAJOR BUG FIXES:

- Updated the auth client to version 0.4.4 to fix the NoSuchMethod error.

2.5.3 VERSION: 0.7.0 (Released 5/5/17)

BACKWARDS INCOMPATIBILITIES:

- It is now required to provide either an object name or an object id when saving an object.

NEW FEATURES:

- `deleteWorkspace` and `undeleteWorkspace` commands have been added to the administration interface.

UPDATED FEATURES / MAJOR BUG FIXES:

- When attempting to save an object with metadata containing a null key or value a more illuminating error is thrown.
- The administration script now uses the authentication service url set in the deploy.cfg file as opposed to a hard coded url.

2.5.4 VERSION: 0.6.0 (Released 12/9/16)

BACKWARDS INCOMPATIBILITIES:

- The `kb|ws...` style of addressing workspaces or objects has been removed.
- A bug allowed workspace names of the form user:X where X is an integer > ~2^32. This style of name is temporarily allowed for backwards compatibility reasons but is deprecated and will be removed in a future release.

NEW FEATURES:

- The `ObjectSpecification` structure now provides a `find_reference_path` field that allows specifying that the permissions for an object should be automatically looked up via a search through the object reference graph.
- The resolved (e.g. all references are absolute) path through the object reference graph from an accessible object to the target object is now returned with `get_objects2` and the new method `get_object_info3`.
- Added a new method, `get_object_info3` that returns the path from an accessible object to the target object, but is otherwise equivalent to `get_object_info_new`. `get_object_info_new` is now deprecated.

- Objects containing a semicolon separated reference path rather than just embedded references can now be saved. If the reference path is valid and the head of the path accessible, the references will be rewritten to the absolute reference of the object at the end of the path.
- Similarly, provenance references can now contain reference paths rather than just single references.

UPDATED FEATURES / MAJOR BUG FIXES:

- The `ObjectSpecification` structure now allows several new ways to provide reference paths into the object graph.
- Fixed a bug where integers > ~2^32 were allowed as workspace and object names.
- Fixed a bug in `register_typespec_copy` where any types in common between the new and previous version of the spec would be unregistered.

2.5.5 VERSION: 0.5.0 (Released 8/12/16)

BACKWARDS INCOMPATIBILITIES:

- The `skip` parameter of `list_objects` has been removed.
- In order to save an object that contains handles to shock nodes, the user must own the shock nodes. Previously, the user only needed read permissions.
- Handle Service version b9de6991b851e9cd8fa9b5012db565f051e0894f+ is now required.
- Handle Manager version 3e60998fc22bb331e51b189ae1b71ebd54e58b90+ is now required.
- Shock version 0.9.6+ is now required.

NEW FEATURES:

- The `status` method now returns JVM memory stats and the status of MongoDB, Shock, and the Handle service and manager (if using the latter three).

UPDATED FEATURES / MAJOR BUG FIXES:

- `clone_workspace` now preserves object IDs from the source workspace such that the object name -> id mapping is identical for both workspaces at the completion of the clone (unless changes are made to the source workspace while the clone is in progress). Due to this change, the maximum object ID returned in a `workspace_info` tuple may be larger than the number of objects in the new workspace. The documentation has been clarified to reflect this.
- `clone_workspace` now prevents the new workspace from being accessed in any way while the clone is in progress.
- `clone_workspace` can now exclude user specified objects from the clone.
- Fixed several bugs where various failures could leave temporary files on disk.
- Fixed a bug where accessing an object with handles to shock nodes anonymously would cause a null pointer error.
- A temporary file is created and deleted at startup to ensure the temporary files directory is readable.
- Fixed a bug where under certain circumstances more data than allowed could be stored in memory or on disk and returned in a get_objects call.
- The authorization URLs used by the server may now be configured.
- All configuration user id / password combinations may now be alternately fulfilled with a token.
- The initialization script now takes a token rather than a user id and password for the shock user account.

2.5.6 VERSION: 0.4.1 (Released 5/27/16)

BACKWARDS INCOMPATIBILIES:

- Java users will need to switch from the `ObjectIdentity` to the `ObjectSpecification` class when calling `getObjectInfoNew`. The interface is a superset of `ObjectIdentity` and so is a simple name swap.
- The text of some error messages has changed.

NEW FEATURES:

- Added the `get_objects2` method. This method combines the functionality of `get_objects`, `get_object_provenance`, `get_object_subset`, and `get_referenced_objects` and as such those methods are deprecated. In particular, a user can now get a subset from a referenced object or get only the provenance from a referenced object. `get_objects2` also allows for returning nulls instead of throwing an error when an object is inaccessible in the same way as `get_object_info_new`.

UPDATED FEATURES / MAJOR BUG FIXES:

- `get_object_info_new` can now follow object references like `get_objects2` and `get_referenced_objects`.
- Fixed an exploit where an attacker, for an arbitrary workspace, could determine the number of objects in that workspace, the number of versions of each object, and whether a particular object name exists in the workspace.
- Added the `custom`, `subactions`, and `caller` fields to `ProvenanceAction`.
- Added original workspace ID to the data returned by `get_objects*` methods.
- Unix epoch times are now accepted and emitted where possible (e.g. not in tuples) as well as string timestamps.
- `list_referencing_object_counts` has been deprecated.

2.5.7 VERSION: 0.4.0 (Released 2/2/16)

BACKWARDS INCOMPATIBILITIES:

- the `list_objects() skip` parameter is now deprecated and will be removed in a future version. Additionally, the `list_objects` method's behavior has changed. `list_objects` is now guaranteed to return either all the remaining objects that match the filters or `limit` objects. `skip` now behaves in an unintuitive way in that the same object may appear in `list_objects` results even when the `skip` parameter setting should ensure that each set of returned objects is disjoint with all the others.
- Module names and type names are now limited to 255 bytes.
- Metadata keys and values are limited to 900B for the total of each pair of key and value.

NEW FEATURES:

- Added `get_permissions_mass` function.
- Added `get_names_by_prefix` function.
- A documentation server now provides all available workspace documentation at the `/docs` endpoint.
- `list_objects` output may now be filtered by minimum and maximum object IDs.

UPDATED FEATURES / MAJOR BUG FIXES:

- Updated for compatibility with Shock 0.9.6 (tests only), 0.9.12, and 0.9.13.

- Removed internal data subsetting (intended for indexing of data contents) code. No plan to use this code and drastically increases database size and codebase complexity. All workspace mongo database `type_[MD5]` collections may be deleted after upgrading.
- Improved logging for the `administer()` method.
- Fixed a bug where mongo connections would not be released when redeploying the server in an already running glassfish instance.
- Fixed a bug where objects from deleted workspaces could be listed in `list_objects` output.
- `get_permissions` no longer requires authentication.
- the admin user specified in the `deploy.cfg` file can no longer be removed by other admins.

2.5.8 VERSION: 0.3.5 (Released 5/15/15)

BUG FIXES:

- Updated auth library dependency that prevented validating user names not in the KBase group, which was preventing sharing with a subset of real and active KBase users.

2.5.9 VERSION: 0.3.4 (Released 4/10/15)

NEW FEATURES:

- Added CLI command for listing properly configured Narratives

UPDATED FEATURES / MAJOR BUG FIXES:

- Updated to the new auth client. Globus APIs changed in a way that broke sharing with multiple users at the same time.
- Added required fields to the deploy.cfg file for user credentials to use when querying user data. These creds must be for an administrator of kbase_users so that all users are visible to the workspace service when attempting to share workspaces.
- Empty strings are now accepted as map keys
- Fixed a NPE when calling list_referencing_object_counts with a non-existent object version
- Fixed a race condition that could occur when operating on an object that's in mid save
- 'strict_maps' and 'strict_arrays' properties are now present in 'get_object_subset' method
- Slashes are now supported in paths used in 'get_object_subset' method

2.5.10 VERSION: 0.3.3 (Released 10/28/14)

NEW FEATURES:

- Object references and types are now logged for many methods.

2.5.11 VERSION: 0.3.2 (Released 10/20/14)

UPDATED FEATURES / MAJOR BUG FIXES:

- The ProvenanceAction data structure now has fields for entering external data sources.

- The workspace client now has streaming mode off by default. To turn it back on, do setStreamingModeOn(true).
- Fixed a bug that would cause calls to the handle service or handle manager to fail every other call if they were not behind nginx and the call frequency was between 1-4s.

2.5.12 VERSION: 0.3.1 (Released 10/1/2014)

UPDATED FEATURES / MAJOR BUG FIXES:

- Fixed a bug where adding an @id annotation to the key of a mapping would result in a minor version increment vs. the expected major version increment.
- Fixed a bug where a bad workspace @id (unparsable, deleted object, etc) with allowed types specified in the typespec would cause a NPE rather than a useful typechecking error.

2.5.13 VERSION: 0.3.0 (Released 9/2/2014)

NEW FEATURES:

- The major change in this release is a major refactoring of the ID handling system. ID handling has been generalized to allow for custom ID handlers per ID type (e.g. the @id [ID_type] annotation).
- The workspace now supports the @id handle annotation, which allows for embedding HandleService handle IDs in workspace objects. When the object is retrieved from the workspace, the user retrieving the object is given read access to any data referenced by handles in the object.
- There is now a limit of 100,000 IDs in objects per save_objects call. IDs duplicated in the same object do not count towards this limit.
- Any IDs extracted from an object are returned in get_objects, get_referenced_objects, get_object_subset, and get_object_provenance.
- The source of a copied object, if visible to the user, is now exposed in the various get_objects* methods.
- New command line scripts added: ws-diff to compare (client side) two workspace objects and ws-typespec-download to automatically download registered typespecs and automatically resolve dependencies.
- Support added for the @metadata ws annotation to automatically extract ws metadata from the object data. String/float/int fields in objects or subobjects can be selected in addition to the length of lists and mappings.
- Support for @range annotation to set limits (inclusive or exclusive) on int and float values.

UPDATED FEATURES / MAJOR BUG FIXES:

- Users with write permissions to a workspace can now view permissions for all users to that workspace.
- X-Forwarded-For and X-Real-IP headers are now taken into account when logging the IP of method calls. Set dont_trust_x_ip_headers=true in deploy.cfg to ignore them.
- Updated timestamp format in ws-list and ws-listobj to display readable local time by default instead of the ISO timestamp.
- get_object_subset no longer generates an error if a selected field or mapping key is not found, which provides better support for optional fields. Errors are still generated if an array element does not exist.

2.5.14 VERSION: 0.2.1 (Released 7/11/14)

NEW FEATURES:

- get_object_provenance returns the object provenance without the data.

- added get_all_type_info and get_all_func_info to return all type/function information registered for a specified module
- a parsed structure of type and function defintions were added to TypeInfo and FuncInfo
- the owner of a module now can determine the released versions of a types and and functions (released version info was added to TypeInfo and FuncInfo)
- Java client now has a method to deactivate SSL certification validation (primarily for use with self-signed certs)

UPDATED FEATURES / MAJOR BUG FIXES:

- the initialization script will no longer allow setting the mongo typedb name to the workspace type db name, and the server will refuse to start up if such is the case.
- configuration of the default URL for the CLI is handled properly; in 0.2.0 the ws-url command needed to be called prior to other commands
- improved documentation and other minor error handling in the CLI
- again allows IRIS deployment of ws-workspace and ws-url
- fixed a bug that could cause date parsing errors on valid incoming date strings
- date strings now may contain 'Z' for the timezone
- kbase user is now configurable for deploy-upstart target
- there is now an option in deploy.cfg to specify the number of times to attempt to contact MongoDB on startup

2.5.15 VERSION: 0.2.0 (Released 5/18/14)

PREAMBLE:

v0.2.0 is a complete rewrite of the data path through the workspace, including type checking, sorting, data extraction, and object retrieval, for the purpose of controlling memory usage.

BACKWARDS INCOMPATIBLITIES:

- deploy.cfg has several new parameters, most of which have acceptable defaults. However temp-dir needs to be set before starting the new version.

NEW FEATURES:

- a new function, list_all_types, returns all the types in the workspace.
- ScriptHelpers workspace library ported to python (from perl) by Mike Mundy.

UPDATED FEATURES / MAJOR BUG FIXES:

- The max object size has been returned to 1GB.
- start_service no longer requires user-env.sh to be sourced.
- Nulls will now pass type checking where an int, float, or string is expected.
- Fixed a bug where get_object_subdata would return the same subdata if two different paths through the same object were specified.
- Command-line interface default URLs are configurable via the makefile.
- ws-workspace and ws-url now work against the User and Job State Service when in IRIS.
- The characters . and - are now allowed in workspace names.
- Parallel GC has been re-enabled.

- Updating a searchable ws or id annotation in a type definition now results in a major version increment instead of a minor version increment.
- Fixed a bug where get_referencing_objects would throw an error if an object has no references.

2.5.16 VERSION: 0.1.6 (Released 3/3/14)

NEW FEATURES:

- Get objects by reference, which allows retrieval of any objects that are referenced by objects to which the user has access.
- A new version of get_object_info, get_object_info_new, allows ignoring errors when listing object information. get_object_info is deprecated in favor of this method.
- Get the number of objects that reference an object via provenance or object- to-object references, including inaccessible objects.

UPDATED FEATURES / MAJOR BUG FIXES:

- Filter list_objects and list_workspace_info by date
- Optionally exclude globally readable objects from list_objects
- list_objects now takes skip and limit parameters and returns at most 10000 objects. list_workspace_objects returns at most 10000 objects.
- A user can reduce their own permissions on any workspace.
- Workspace and object names can now be up to 255 characters in length.
- Workspace mod dates are now updated on a save/copy/revert/delete/rename of an object.
- Fixed a bug that caused object checksums to be calculated incorrectly. Note that any checksums calculated before this version are incorrect.
- Fixed a bug where trying to copy an object to an object with a version > than the maximum existing version would fail. The incoming copy target version number should be ignored.
- Fixed a bug where trying to copy an object to a deleted object would fail.
- Clarified some exceptions / error messages.

2.5.17 VERSION: 0.1.5 (Released 2/5/14)

Hotfix to use updated auth libs with 60d token lifetime.

2.5.18 VERSION: 0.1.4 (Released 1/30/14)

NEW FEATURES:

- Get the version of the workspace server.
- Set metadata on a workspace and search workspaces by metadata.

UPDATED FEATURES / MAJOR BUG FIXES:

- On startup the WSS attempts to create a node in shock to test for shock misconfiguration (shock client change)

2.5.19 VERSION: 0.1.3 (Released 1/24/14)

UPDATED FEATURES / MAJOR BUG FIXES:

- Fixed a bug where get_module_info and get_type_info reported removed types.
- Scripts now allow IDs or object references to be used in place of object and workspace names.

2.5.20 VERSION: 0.1.2 (Released 1/23/14)

Hotfix release to disallow integer object and workspace names.

2.5.21 VERSION: 0.1.1 (Released 1/21/14)

BACKWARDS INCOMPATIBILITIES:

- The maximum object size is temporarily limited to 200MB.
- The maximum JSON string size received by the server is temporarily limited to 250MB.

NEW FEATURES:

- Add owners to modules so that multiple users can upload typespecs.
- Option to list only deleted objects or workspaces.
- Filter objects or workspaces list by permission level.
- Filter workspaces list by owner.
- Filter object list by the person who saved the object.
- Filter object list by user metadata.
- Return a list of objects that reference another object, either in the object data or the provenance data.

UPDATED FEATURES / MAJOR BUG FIXES:

- Module owners can now see unreleased modules and types.
- Turned off parallel garbage collection - was locking the server when processing large objects.
- Fixed bug in WS ID relabeling in values of mappings when keys contain forward slash character
- Retrieving subset of an object that includes an array element out of the array index range now generates an error instead of returning a subset with null values in the array
- First error encountered during type checking halts type checking, meaning that only the first error is shown to you even if multiple errors exist

2.5.22 VERSION: 0.1.0 (Released 1/9/2014)

PREAMBLE:

0.1.0 is a complete rewrite of the workspace service and thus has many changes to the API. A function change list is below.

NEW FEATURES:

- The WSS is configurable to save TOs in MongoDB/GridFS or Shock.
- Load, compile, and view KIDL typespecs.

- Objects are type checked against a KIDL typespec before saving.
- Save provenance information with an object.
- References to other workspace objects in a TO or TO provenance are confirmed accessible and type checked before saving.
- A list of references from a TO or TO provenance to other workspace objects is saved and retrievable.
- Hide objects. Hidden objects, by default, do not appear in the list_* methods.
- Lock a workspace, freezing it permanently. Locked, publicly readable workspaces are published.
- Workspaces and objects have a permanent autoincrementing ID as well as a mutable name. An object may be addressed by any combination of the workspace and object name or id plus a version number, or the KBase ID kb|ws.[workspace id].obj.[object id].ver.[object version].
- Workspaces may have a <1000 character description.
- Workspace names may be prefixed by the user's username and a colon. This provides a unique per user namespace for workspace names.
- Return only a user specified subset of an object.

UPDATED FEATURES / MAJOR BUG FIXES:

- Many methods now operate on multiple objects rather than one object per method call.
- list_objects can list objects from multiple workspaces at once.
- Rename an object or workspace.

FUNCTION CHANGE LIST:

Deprecated functions, and their replacement

get_workspacemeta -> get_workspace_info
get_objectmeta -> get_object_info
save_object -> save_objects
get_object -> get_objects
list_workspaces -> list_workspace_info
list_workspace_objects -> list_objects

Functions with an altered api. Please see the API documentation for details

create_workspace
clone_workspace
get_objects
copy_object
revert_object
object_history -> get_object_history
set_global_workspace_permissions -> set_global_permission
set_workspace_permissions -> set_permissions
get_workspacepermissions -> get_permissions
delete_workspace -> delete_workspace and undelete_workspace
delete_object -> delete_objects and undelete_objects

Removed functions

move_object -> use rename_object or copy_object and delete_objects
has_object -> use get_object_info
delete_object_permanently
add_type -> various new functions below
get_types -> various new functions below
remove_type
load_media_from_bio
import_bio
import_map
queue_job -> AWE and / or the UserJobStateService
set_job_status -> AWE and / or the UserJobStateService
get_jobs -> AWE and / or the UserJobStateService
get_object_by_ref
save_object_by_ref
get_objectmeta_by_ref
get_user_settings -> UserJobStateService
set_user_settings -> UserJobStateService

New functions

get_object_subset
get_workspace_description
set_workspace_description
lock_workspace
rename_workspace
rename_object
hide_objects
unhide_objects
request_module_ownership
register_typespec
register_typespec_copy
release_module
list_modules
list_module_versions
get_module_info
get_jsonschema
translate_from_MD5_types
translate_to_MD5_types
get_type_info
get_func_info
administer

2.5.23 VERSION: 0.0.5 (Released 11/19/2013)

NEW FEATURES:

- Type compiler provided embedded authorization works
- Connect to mongodb databases requiring authorization
- Optionally exclude world readable workspaces from the output of list_workspaces()

UPDATED FEATURES / MAJOR BUG FIXES:

- Authentication is required for all writes, including workspace creation. The 'public' user is now no different from any other user
- Workspace default permissions are now limited to none and read only
- A user must have at least read access to a workspace to get its metadata
- Only the user's own permission level is now returned by get_workspacepermissions() if a user has read or write access to a workspace
- Only the workspace's owner can change the owner's permissions
- Type names are now limited to ascii alphanumeric characters and _
- Object names are now limited to ascii alphanumeric characters and .|_-
- Object names must now be unique per workspace, even if the objects are different types
- Object and workspace names may not be integers
- Removed one of the two python clients in lib/, as it was not being updated on a make while the other was

2.5.24 VERSION: 0.0.4 (Released 8/13/2013)

NEW FEATURES:

- Connect to mongodb databases requiring authorization
- get_objects() method

2.5.25 VERSION: 0.0.3 (Released 1/1/2012)

NEW FEATURES:

- Added functions to manage the addition and removal of types.
- Added functions to handle job management to support running jobs on local clusters
- Added "instance" argument to "get_object" to enable users to access all object instances
- Created a complete set of command line scripts for interacting with workspace

UPDATED FEATURES / MAJOR BUG FIXES:

- Added ability to retrieve specific instances of objects
- Fixed bug in deletion of workspaces
- Fixed bug in object reversion
- Fixed bug in object retrieval
- Fixed bug in management of persistant state in workspace

2.5.26 VERSION: 0.0.2 (Released 11/30/2012)

NEW FEATURES:

- This is the first public release of the Workspace Services.
- adjusted functions to accept arguments as a hash instead of an array
- added ability to provide authentication token in input arguments

2.5.27 VERSION: 0.0.1 (Released 10/12/2012)

NEW FEATURES:

- This is the first internal release of the Workspace Service, all methods are new.

2.6 Documentation TODO list

Todo: Add js and py documentation when the typecompiler / kb-sdk generates it

(The original entry is located in /home/apasha/dev/kbase/workspace_deluxe/docsource/api.rst, line 11.)

Todo: Build and initialization instructions for the Perl client. If this can be done without the KBase runtime & dev_container that'd be ideal.

(The original entry is located in /home/apasha/dev/kbase/workspace_deluxe/docsource/buildinitclient.rst, line 144.)

Todo: Build (probably not needed) and initialization instructions for the Javascript client.

(The original entry is located in /home/apasha/dev/kbase/workspace_deluxe/docsource/buildinitclient.rst, line 151.)

Todo: Find or create some user-facing bugs

(The original entry is located in /home/apasha/dev/kbase/workspace_deluxe/docsource/knownuserbugs.rst, line 4.)

Todo: Update this document to use the kb-sdk tools.

(The original entry is located in /home/apasha/dev/kbase/workspace_deluxe/docsource/typedobjects.rst, line 92.)