



Java SDK

Overview

The VersionOne Java SDK is a library built using Java that allows object-oriented access to the VersionOne API, which is inherently a REST-based web service. Using the SDK you can query for simple or complex sets of information, update information, and execute system-defined operations, without having to construct HTTP requests and responses or deal with parsing the data contained in the HTTP responses. The SDK serves as a wrapper to the VersionOne API, eliminating the need to code the infrastructure necessary for direct handling of HTTP requests and responses.

Practically all data in VersionOne is stored in the form of [assets](#), which have [attributes](#). Each asset is classified by an [asset type](#), which describes a number of [attribute definitions](#), [operations](#), and possibly a [relationship](#) to another asset type. A list of all the types within VersionOne, including their attributes and operations, can be obtained using the [Meta API](#).

Simple queries can request a single VersionOne asset with several attributes. Complex queries can request multiple assets meeting certain criteria, have the results sorted in a particular way, and even ask for a portion (or "page") of the overall results. When constructing your queries, you must use the system name for the asset type you would like to retrieve. This is true whether using the API directly or the SDK.

In terms of [API authentication](#), the SDK supports using [Basic \(username and password\)](#), [Windows Integrated \(NTLM\)](#), [Access Tokens](#), and [OAuth2](#).



System Requirements

The current release of the Java SDK requires JRE 1.8 or higher, and VersionOne 8.0 or higher (though some features are only available with newer releases).

Getting the Java SDK

There are two ways that you can get the SDK: by using [Maven](#) to import the SDK and its dependencies from the [Maven Central Repository](#), or by downloading the SDK and its associated dependencies from the [VersionOne AppCatalog](#) and setting a reference to it manually from within your project.

The recommended approach is to use Maven as it will take care of all the dependencies for you, plus make it easier to update the SDK as new versions become available.

Setting a reference using Maven

Assuming that you are using Eclipse as your IDE, use the following steps to add the Java SDK to your project using Maven:

1. Launch Eclipse and open your project.
2. Edit the POM file and add the following dependency with the actual SDK version number that you want to reference:

```
<dependency>
  <groupId>com.versionone</groupId>
  <artifactId>VersionOne.SDK.Java.APIClient</artifactId>
  <version>XX.X.X</version>
</dependency>
```

Setting a reference manually

Assuming that you are using Eclipse as your IDE, use the following steps to manually add the Java SDK to your project:

1. Download the latest stable build of the SDK from the AppCatalog page.
2. Save the ZIP file to a known location on your hard drive.
3. Unzip the SDK.
4. Launch Eclipse and open your project.
5. In the Package Explorer, right click on the project name then select **Build Path**, then **Add External Archives**.
6. Navigate to the directory that you downloaded the SDK to and select the VersionOne.SDK.Java.APIClient-XX.X.X.jar file, then click **Open**.

While the Java SDK is fully supported by VersionOne, it is also open-sourced. You can get the source code for the SDK from this GitHub repository: <https://github.com/versionone/VersionOne.SDK.Java.APIClient>



Using the Java SDK

Once you properly set a reference to the SDK within your project, using it is simply a matter of making a connection to VersionOne then writing the code to query, update, and create VersionOne assets, or execute operations against them.

The following topics discuss the major features of the SDK and demonstrate how you can make use of them:

- [Creating a Connection](#)
- [Querying Assets](#)
- [Querying Asset History](#)
- [Querying Configurations and Localizations](#)
- [Creating Assets](#)
- [Updating Assets](#)
- [Executing Operations](#)
- [Executing Pass-Through Queries](#)
- [Working with Attachments and Images](#)

Getting Help

While we strive to make the SDK as easy to use as possible, you may still occasionally need some help, and there are a few different ways you can get it:

- [Code Samples](#): A growing list of recipes for working with the VersionOne API. Check here first to see if an answer already exists.
- [StackOverflow](#): For asking question of the VersionOne Developer Community.
- [VersionOne Support](#): Our support team is well versed in the VersionOne API, including accessing it via the SDK.
- [VersionOne Technical Services](#): A paid support offering, this team provides API training, development pairing, and complete custom development services.



Creating Assets

When you create a new asset with the Java SDK, you need to specify the context of another asset that will be the parent. For example, if you create a new Story asset you can specify which Scope (project) it should be created in.

Prior to creating an asset in VersionOne, you must first instantiate a V1Connector and Services object:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8IKLk+PnyFxEWVX5KI2u6Jk=")
    .build();

IServices services = new Services(connector);
```

Creating a New Asset

This example shows how to create a Story asset in the context of a Scope with ID 1012:

```
Oid projectId = services.getOid("Scope:0");
IAssetType storyType = services.getMeta().getAssetType("Story");
Asset newStory = services.createNew(storyType, projectId);
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
newStory.setAttributeValue(nameAttribute, "My New Story");
services.save(newStory);

System.out.println(newStory.getOid().getToken());
```



```
System.out.println(newStory.getAttribute(storyType.getAttributeDefinition("Scope")).getValue());
System.out.println(newStory.getAttribute(nameAttribute).getValue());
```

```
/***** OUTPUT *****/
```

```
Story:7617:9243
```

```
Scope:0
```

```
My New Story
```

```
*****/
```



Creating a Connection

Connecting to the VersionOne API via the Java SDK involves determining the URL of your VersionOne instance, determining the [API authentication](#) type that you want use, determining your proxy credentials (if you use one), and then building a connection object. To build a connection object you'll use the **V1Connector** object which is implemented using a fluent builder interface.

Here's an example of how to use the **V1Connector** with an access token:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKLk+PnyFkEWVX5KI2u6Jk=")
    .build();
```

A few things to point out about the **V1Connector**:

- The [Server Base URI](#) is the URL that you use for your VersionOne instance and is typically in the form of "http(s)://server name/instance name".
- The [user agent header](#) is used to pass the name and version number of your application to the API which can help with log analysis should there be an issue.
- The **build** method is the builder's terminating method and returns the **V1Connector** object which you will then pass to other objects when performing actions with the API.

The **V1Connector** is a new connector released with the 15.0.0.0 version of the Java SDK, and it replaces the legacy **V1APIConnector**. The legacy connector is still available for use within the SDK but has been marked for deprecation. It will be removed in a future release.



Connecting with Basic Authentication

When using [Basic authentication](#), you use the **withUsernameAndPassword** method, passing in the username and password of the VersionOne member account that you want to connect with:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withUsernameAndPassword("username", "password")
    .build();
```

Connecting with Windows Integrated Authentication

When using [Windows Integrated Authentication](#), you use the **withWindowsIntegrated** method. If you want to use the SDK with the currently logged in user's account, you do not need to pass any parameters:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withWindowsIntegrated()
    .build();
```

Unlike the [.NET SDK](#), the Java SDK does not support using Windows Integrated Authentication with specific user credentials, you can only use the credentials of the currently logged in user.

Connecting with Access Tokens

When using [Access Token Authentication](#), you use the **withAccessToken** method, passing in the access token associated with the VersionOne member account that you want to connect with:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKLk+PnyFxEWVX5KI2u6Jk=")
    .build();
```

When trying to use an access token when your VersionOne instance has been configured to use Windows Integrated Authentication, you will need to use the special **useOAuthEndpoints** method of the connector. This method allows the connector to bypass NTLM and authenticate directly to the API with an access token.

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKLk+PnyFxEWVX5KI2u6Jk=")
    .useOAuthEndpoints()
    .build();
```



Connecting with OAuth2 Tokens

As of Winter 2016 release notes our intent is to sunset OAuth2 in favor of Access Tokens.

When using [OAuth2 Authentication](#), you use the **withOAuth2Token** method, passing in the OAuth2 access token associated with the VersionOne member account that you want to connect with:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withOAuth2Token("AAEAAJa0PvSSjkBYffqU2f5oPCFLimIhiiQaiM04VO-5p7NmIm0W9bqM59...")
    .build();
```

The Java SDK only supports connecting with a valid OAuth2 access token, it does not do the full authorization of the token, nor does it refresh the token. If you wish to use OAuth2 as your authentication method, you may want to use a third-party library like [Apache OLTU](#) or the [Google OAuth Client Library](#) to assist with generating and refreshing your OAuth2 access tokens.

Connecting with a Proxy

If you are using a proxy in your environment, you can use the **withProxy** method, passing in a **ProxyProvider** object hydrated with the URL, username, and password used to authenticate with the proxy:

```
ProxyProvider proxyProvider = new ProxyProvider(new URI("proxyURL"), "proxyUsername", "proxyPassword");

V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8IKLk+PnyFxEWVX5KI2u6Jk=")
    .withProxy(proxyProvider)
    .build();
```

You cannot use a proxy when connecting with Windows Integrated Authentication.



Creating a Services Object

Once you have created a **V1Connector** instance, you'll need to create an instance of the **Services** class, which is the primary object that you will use to perform actions with the VersionOne API. There are three constructors for the **Services** class. All of them require a **V1Connector** instance, however.

Creating a V1Connector Instance

Here's a simple example of creating a **V1Connector** instance. This example uses an [Access Token](#). For other types of connections, see the [Creating a Connection](#) topic.

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8IKLk+PnyFxEWVX5KI2u6Jk=")
    .build();
```

Services(V1Connector v1Connector)

To create an instance of **Services** that creates an **IMetaModel** instance for you automatically, just use the one-parameter version of the constructor, passing in your **V1Connector** instance. By default, the connector will not pre-load the [Meta](#) information. If you need to do that, see the next overload.

```
IServices services = new Services(connector);
```



Services(V1Connector connector, boolean preLoadMeta)

If you'd like to force the connector to pre-load the [Meta](#) information before you work with the Services instance, call this version of the constructor and specify true for the second parameter. Note that this can cause the start-up time for your code to take a long time, but it is ultimately faster if you do a significant number of queries against a large variety of Asset types or Attributes. If you work with a smaller variety, then allowing the connector to dynamically fetch [Meta](#) is a better approach.

```
| IServices services = new Services(connector, true);
```

Services(V1Connector connector, IMetaModel metaModel)

Though it would rarely be needed for most purposes, if you want complete control over creating the **IMetaModel** instance that you configure your **Services** instance with, you can use the third overload of the **Services** constructor. One scenario for when you would use this is if you wish to create your own implementation of the **IMetaModel** interface instead of relying upon the library-provided **MetaModel** class. You might do this if you wanted to cache definitions on disk instead of reaching out to the instance for every call.

The following sample produces the same result as the previous constructor example:

```
| // Using the library-provided MetaModel class's constructor
| IMetaModel metaModel = new MetaModel(connector, false); // false indicates to NOT preload meta
| IServices services = new Services(connector, metaModel);
```

If you do implement your own implementation of the **IMetaModel** interface, then your code would look something like this:

ParseError: EOF expected (click for details)

```
| Callstack:
|   at (VersionOne_Connect/Developer_Library/Get_an_SDK/Java_SDK/Creating_a_Services_Object), /content/
|   body/div[2]/div[2]/div[2]/pre, line 3, column 12
```





Executing Operations

An [operation](#) is an action that is executed against a single [asset](#). For example, to delete an asset you must execute the Delete operation on the asset. To close or inactivate a asset, you use the Inactivate operation.

You can use the [Meta API](#) to determine the operations that a particular asset supports

Prior to executing an operation against an asset in VersionOne, you must first instantiate a **V1Connector** and **Services** object:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKLk+PnyFxEWVX5KI2u6Jk=")
    .build();

IServices services = new Services(connector);
```

In versions of the SDK prior to the 15.0.0.0 release, you would also have to instantiate a connector for the **MetaModel** object. However, starting with the 15.0.0.0 release, that is no longer necessary.

The **MetaModel** object is now available from the **getMeta** method of the **Services** object. For more advanced **Services** constructor overloads, see the [Creating a Services Object](#) topic.



Closing an Asset

This example shows how to close an asset using the **executeOperation** method of the **Services** object. Note the use of the **getOperation** method to get the operation to execute:

```
IOperation closeOperation = services.getMeta().getOperation("Story.Inactivate");
Oid closeID = services.executeOperation(closeOperation, services.getOid("Story:7618"));

Query query = new Query(closeID.getMomentless());
IAttributeDefinition assetState = services.getMeta().getAttributeDefinition("Story.AssetState");
query.getSelection().add(assetState);
QueryResult result = services.retrieve(query);
Asset closeStory = result.getAssets()[0];
AssetState state = AssetState.valueOf(((Integer) closeStory.getAttribute(assetState).getValue()).intValue());

System.out.println(closeStory.getOid());
System.out.println(state.toString());

/***** OUTPUT *****/
Story:7618
Closed
*****/
```

Reopening an Asset

This example shows how to reopen an asset using the **executeOperation** method of the **Services** object:

```
IOperation closeOperation = services.getMeta().getOperation("Story.Reactivate");
Oid closeID = services.executeOperation(closeOperation, services.getOid("Story:7618"));

Query query = new Query(closeID.getMomentless());
IAttributeDefinition assetState = services.getMeta().getAttributeDefinition("Story.AssetState");
query.getSelection().add(assetState);
QueryResult result = services.retrieve(query);
Asset closeStory = result.getAssets()[0];
AssetState state = AssetState.valueOf(((Integer) closeStory.getAttribute(assetState).getValue()).intValue());

System.out.println(closeStory.getOid());
System.out.println(state.toString());

/***** OUTPUT *****/
Story:7618
Active
*****/
```



Deleting an Asset

This example shows how to delete an asset using the **executeOperation** method of the **Services** object:

```
IOperation deleteOperation = services.getMeta().getOperation("Story.Delete");
Oid deletedOID = services.executeOperation(deleteOperation, services.getOid("Story:7618"));

try {
    Query query = new Query(deletedOID.getMomentless());
    @SuppressWarnings("unused")
    QueryResult result = services.retrieve(query);
} catch (ConnectionException e) {
    System.out.println(String.format("%s has been deleted", deletedOID.getMomentless()));
}

/***** OUTPUT *****/
Story:7618 has been deleted
*****/
```

The **Delete** operation returns the [Oid](#), with the new [Moment](#), of the deleted asset. Future current info queries will automatically exclude deleted assets from results.



Executing Pass-Through Queries

In addition to providing object model-like access to the VersionOne Data API, the Java SDK also provides a way to execute queries using the Query API. The Query API provides read-only access to VersionOne data, and allows you to submit hierarchical queries in a JSON or YAML format. In addition, data returned from the Query API is in a JSON format.

Executing a JSON Query

In this example, a JSON query is used to get all Story assets with an estimate greater than ten:

```
String query =
    "{" +
    "  \"from\": \"Story\", \" +
    "  \"select\": [\"Name\", \"Number\"]" +
    "}";

String result = services.executePassThroughQuery(query);
System.out.println(result);
```

This is an example of the raw JSON data that is returned:

```
[
  [
    {
      "_oid": "Story:6555",
      "Name": "Test Story on Scope:6527 - Name attribute"
    },
  ],
]
```



```
{
  "_oid": "Story:6588",
  "Name": "Test Story Scope:6527 Query filter with multiple attributes"
}
]
```

Executing a YAML Query

In this example, a YAML query is used to get all Story assets with an estimate greater than ten, the raw JSON that is returned is the same as when submitting the query in a JSON format:

```
String query = "from: Story\n" +
  "select:\n" +
  "  - Name\n" +
  "  - Number\n";

String result = services.executePassThroughQuery(query);
System.out.println(result);
```





Querying Assets

One of the most fundamental, if not the most common, things that you can do with the Java SDK is to query VersionOne for information about the assets that it contains. To do so, all you need is a valid V1Connector object and an instance of the Services object, which is the primary object that you will use to perform actions with the VersionOne API. Once you have those, you can then write queries to access data for just about any asset contained within VersionOne. Here's a quick example of instantiating a V1Connector and Services object: V1Connector connector = V1Connector .withInstanceUrl("<Server Base URI>") .withUserAgentHeader("AppName", "1.0") .withAccessToken("1.rWM8IKLk+PnyFxEWVX5KI2u6Jk=") .build(); IServices services = new Services(connector); In versions of the SDK prior to the 15.0.0.0 release, you would also have to instantiate a connector for the MetaModel object. However, starting with the 15.0.0.0 release, that is no longer necessary. The MetaModel object is now available from the getMeta method of the Services object. For more advanced Services constructor overloads, see the Creating a Services Object topic.

Quick Start

Here's a quick example of instantiating a V1Connector and Services object:

```
V1Connector connector = V1Connector .withInstanceUrl("<Server Base URI>")
.withUserAgentHeader("AppName", "1.0")
.withAccessToken("1.rWM8IKLk+PnyFxEWVX5KI2u6Jk=")
.build();
IServices services = new Services(connector);
```



In versions of the SDK prior to the 15.0.0.0 release, you would also have to instantiate a connector for the MetaModel object. However, starting with the 15.0.0.0 release, that is no longer necessary. The MetaModel object is now available from the getMeta method of the Services object. For more advanced Services constructor overloads, see the [Creating a Services Object](#) topic.

Querying a Single Asset

In this example, the asset will have its [OID](#) populated, but will not have any other [attributes](#) populated. This is to minimize the size of the data sets returned. The next example shows how to ask for an asset with specific attributes populated:

```
Oid memberId = services.getOid("Member:20");
Query query = new Query(memberId);
QueryResult result = services.retrieve(query);
Asset member = result.getAssets()[0];

System.out.println(member.getOid().getToken());

/***** OUTPUT *****/
Member:20
*****/
```

Querying an Asset for Specific Attributes

This example shows how to retrieve an asset with specific attributes using the **getSelection** method of the **Query** object:

```
Oid memberId = services.getOid("Member:20");
Query query = new Query(memberId);
IAttributeDefinition nameAttribute = services.getMeta().getAttributeDefinition("Member.Name");
IAttributeDefinition emailAttribute = services.getMeta().getAttributeDefinition("Member.Email");
query.getSelection().add(nameAttribute);
query.getSelection().add(emailAttribute);
QueryResult result = services.retrieve(query);
Asset member = result.getAssets()[0];

System.out.println(member.getOid().getToken());
System.out.println(member.getAttribute(nameAttribute).getValue());
System.out.println(member.getAttribute(emailAttribute).getValue());

/***** OUTPUT *****/
Member:20
Administrator
admin@company.com
*****/
```



Query for a List of Assets

This example shows how to retrieve specific attributes for all the Stories contained within VersionOne:

```
IAssetType storyType = services.getMeta().getAssetType("Story");
Query query = new Query(storyType);
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
IAttributeDefinition estimateAttribute = storyType.getAttributeDefinition("Estimate");
query.getSelection().add(nameAttribute);
query.getSelection().add(estimateAttribute);
QueryResult result = services.retrieve(query);

for (Asset story : result.getAssets()) {
    System.out.println(story.getOid().getToken());
    System.out.println(story.getAttribute(nameAttribute).getValue());
    System.out.println(story.getAttribute(estimateAttribute).getValue());
    System.out.println();
}

/***** OUTPUT *****/
Story:1083
View Daily Call Count
5

Story:1554
Multi-View Customer Calendar
1 ...
*****/
```

Depending on your security role, you may not be able to see all the Story assets in the entire system.

Querying with Filtering on a Single Attribute

This example shows how to query using a **FilterTerm** with the **setFilter** method of the **Query** object to filter the results that are returned. This query will retrieve only Task assets with a **ToDo** value of zero:

```
IAssetType taskType = services.getMeta().getAssetType("Task");
Query query = new Query(taskType);
IAttributeDefinition nameAttribute = taskType.getAttributeDefinition("Name");
IAttributeDefinition todoAttribute = taskType.getAttributeDefinition("ToDo");
query.getSelection().add(nameAttribute);
query.getSelection().add(todoAttribute);

FilterTerm todoTerm = new FilterTerm(todoAttribute);
todoTerm.equal(0);
query.setFilter(todoTerm);
QueryResult result = services.retrieve(query);

for (Asset task : result.getAssets()) {
```



```

        System.out.println(task.getOid().getToken());
        System.out.println(task.getAttribute(nameAttribute).getValue());
        System.out.println(task.getAttribute(todoAttribute).getValue());
        System.out.println();
    }

```

/***** OUTPUT *****/

```

Task:1153
Code Review
0

```

```

Task:1154
Design Component
0 ...

```

*****/

Querying with Filtering on Multiple Attributes

This example shows how to group multiple filter terms using the **GroupFilterTerm** and **FilterTerm** objects, then setting the filter for the query using the **setFilter** method of the **Query** object. This query will retrieve only Defect assets in the base system project with a ToDo value of zero:

```

Oid projectId = services.getOid("Scope:0");
IAssetType assetType = services.getMeta().getAssetType("Defect");

Query query = new Query(assetType);
IAttributeDefinition projectAttribute = assetType.getAttributeDefinition("Scope");
IAttributeDefinition todoAttribute = assetType.getAttributeDefinition("ToDo");
query.getSelection().add(projectAttribute);
query.getSelection().add(todoAttribute);

```

```

FilterTerm projectTerm = new FilterTerm(projectAttribute);
projectTerm.equal(projectId);
FilterTerm todoTerm = new FilterTerm(todoAttribute);
todoTerm.equal(0);

```

```

GroupFilterTerm groupFilter = new AndFilterTerm(projectTerm, todoTerm);
query.setFilter(groupFilter);

```

```

QueryResult result = services.retrieve(query);
for (Asset task : result.getAssets()) {
    System.out.println(task.getOid().getToken());
    System.out.println(task.getAttribute(projectAttribute).getValue());
    System.out.println(task.getAttribute(todoAttribute).getValue());
    System.out.println();
}

```

/***** OUTPUT *****/

```

Defect:37396
Scope:0
0.0

```



Defect:39675
Scope:0
0.0
*****/

Querying with Searching

This example shows how to use the **setFind** method of the **Query** object to search for text. This query will retrieve all Story assets with the word "Urgent" in their name:

```
IAssetType requestType = services.getMeta().getAssetType("Story");
Query query = new Query(requestType);
IAttributeDefinition nameAttribute = requestType.getAttributeDefinition("Name");
query.getSelection().add(nameAttribute);

AttributeSelection selection = new AttributeSelection();
selection.add(nameAttribute);
query.setFind(new QueryFind("Urgent", selection));
QueryResult result = services.retrieve(query);

for (Asset request : result.getAssets())
{
    System.out.println(request.getOid().getToken());
    System.out.println(request.getAttribute(nameAttribute).getValue());
    System.out.println();
}

/***** OUTPUT *****/
Story:1195
Urgent! Filter by owner
*****/
```

Querying with Sorting

This example shows how to use the **getOrderBy** method of the **Query** object to sort the results. This query will retrieve all Story assets sorted by increasing Estimate:

```
IAssetType storyType = services.getMeta().getAssetType("Story");
Query query = new Query(storyType);
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
IAttributeDefinition estimateAttribute = storyType.getAttributeDefinition("Estimate");
query.getSelection().add(nameAttribute);
query.getSelection().add(estimateAttribute);
query.getOrderBy().minorSort(estimateAttribute, Order.Ascending);
QueryResult result = services.retrieve(query);

for (Asset story : result.getAssets()) {
```



```

    System.out.println(story.getOid().getToken());
    System.out.println(story.getAttribute(nameAttribute).getValue());
    System.out.println(story.getAttribute(estimateAttribute).getValue());
    System.out.println();
}

/***** OUTPUT *****/
Story:1073
Add Order Line
1
Story:1068
Update Member
2 ...
*****/

```

There are two methods you can call on the **OrderBy** object to sort your results: **minorSort** and **majorSort**. If you are sorting by only one field, it does not matter which one you use. If you want to sort by multiple fields, you need to call either **minorSort** or **majorSort** multiple times. The difference is that each time you call **minorSort**, the parameter will be added to the end of the OrderBy statement. Each time you call **majorSort**, the parameter will be inserted at the beginning of the OrderBy statement.

Querying with Paging

This example shows how to retrieve a "page" of query results by using the **getPaging** method of the **Query** object. This query will retrieve the first 3 Story assets:

```

IAssetType storyType = services.getMeta().getAssetType("Story");
Query query = new Query(storyType);
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
IAttributeDefinition estimateAttribute = storyType.getAttributeDefinition("Estimate");
query.getSelection().add(nameAttribute);
query.getSelection().add(estimateAttribute);
query.getPaging().setPageSize(3);
query.getPaging().setStart(0);
QueryResult result = services.retrieve(query);

for (Asset story : result.getAssets()) {
    System.out.println(story.getOid().getToken());
    System.out.println(story.getAttribute(nameAttribute).getValue());
    System.out.println(story.getAttribute(estimateAttribute).getValue());
    System.out.println();
}

/***** OUTPUT *****/
Story:1063
Logon
2

Story:1064
Add Customer Details
2

```



```
Story:1065
Add Customer Header
3
*****/
```

The **setPageSize** method shown asks for 3 items, and the **setStart** method indicates to start at 0. The next 3 items can be retrieve with **setPageSize=3, setStart=3**.

Querying with Downcasting

This example shows how to use a [downcast](#) to select the Name attribute of all Test assets associated with a specific Story:

```
Oid assetOID = services.getOid("Story:7608");
IAssetType assetType = services.getMeta().getAssetType("Story");
Query query = new Query(assetOID);
IAttributeDefinition nameAttribute = assetType.getAttributeDefinition("Children:Test.Name");
query.getSelection().add(nameAttribute);
QueryResult result = services.retrieve(query);

for (Object value : result.getAssets()[0].getAttribute(nameAttribute).getValues()) {
    System.out.println(value);
}

/**** OUTPUT ****
Test #1
Test #2
*****/
```

Querying with Functions

This example shows how to use a function to sum the DetailEstimate values for all Task assets associated with a specific Story:

```
Oid assetOID = services.getOid("Story:7608");
IAssetType assetType = services.getMeta().getAssetType("Story");
Query query = new Query(assetOID);
IAttributeDefinition sumAttribute = assetType.getAttributeDefinition("Children:Task.DetailEstimate.@Sum");
query.getSelection().add(sumAttribute);
QueryResult result = services.retrieve(query);

System.out.println(result.getAssets()[0].getAttribute(sumAttribute).getValue());

/**** OUTPUT ****
20.0
*****/
```



Querying Asset History

It is often useful to query VersionOne for the history of a particular asset. This can be useful for reporting on how an asset has changed over time, or to see who has changed it and the moment the change occurred. Querying for asset history is similar to querying for current asset data, the difference being that you set the optional Historical parameter of the Query object to "true".

Querying the History of a Single Asset

This example shows how to retrieve the history of the Member asset with ID 1000:

```
IAssetType memberType = services.getMeta().getAssetType("Member");
Query query = new Query(memberType, true);
IAttributeDefinition idAttribute = memberType.getAttributeDefinition("ID");
IAttributeDefinition changeDateAttribute = memberType.getAttributeDefinition("ChangeDate");
IAttributeDefinition emailAttribute = memberType.getAttributeDefinition("Email");
query.getSelection().add(changeDateAttribute);
query.getSelection().add(emailAttribute);
FilterTerm idTerm = new FilterTerm(idAttribute);
idTerm.equal("Member:1000");
query.setFilter(idTerm);
QueryResult result = services.retrieve(query);

for (Asset member : result.getAssets()) {
    System.out.println(member.getId().getToken());
    System.out.println(member.getAttribute(changeDateAttribute).getValue());
    System.out.println(member.getAttribute(emailAttribute).getValue());
    System.out.println();
}
```



```

}

/***** OUTPUT *****/
Member:1000:105
4/2/2015 1:22:03 PM
andre.agile@company.com

Member:1000:101
3/29/2015 4:10:29 PM
andre@company.net
*****/

```

As demonstrated in the example above, to create a history query, you provide a boolean value of "true" to the second argument of the **Query** object constructor.

Querying the History of Multiple Assets

This example shows how to retrieve history for all Member assets:

```

IAssetType memberType = services.getMeta().getAssetType("Member");
Query query = new Query(memberType, true);
IAttributeDefinition changeDateAttribute = memberType.getAttributeDefinition("ChangeDate");
IAttributeDefinition emailAttribute = memberType.getAttributeDefinition("Email");
query.getSelection().add(changeDateAttribute);
query.getSelection().add(emailAttribute);
QueryResult result = services.retrieve(query);

for (Asset member : result.getAssets()) {
    System.out.println(member.getOid().getToken());
    System.out.println(member.getAttribute(changeDateAttribute).getValue());
    System.out.println(member.getAttribute(emailAttribute).getValue());
    System.out.println();
}

/***** OUTPUT *****/
Member:1010:106
4/2/2015 3:27:23 PM
tammy.coder@company.com

Member:1000:105
4/2/2015 1:22:03 PM
andre.agile@company.com

Member:1000:101
3/29/2015 4:10:29 PM
andre@company.net
*****/

```

Again, the response is a list of historical assets. There will be multiple **Asset** objects returned for an asset that has changed previously.



Querying Asset History "as of" a Specific Point in Time

This example shows how to use the **setAsOf** method of the **Query** object to retrieve data as it existed at some point in time. This query finds the version of each Story asset as it existed seven days ago:

```
IAssetType storyType = services.getMeta().getAssetType("Story");
Query query = new Query(storyType, true);
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
IAttributeDefinition estimateAttribute = storyType.getAttributeDefinition("Estimate");
query.getSelection().add(nameAttribute);
query.getSelection().add(estimateAttribute);

Calendar c = Calendar.getInstance();
c.add(Calendar.DAY_OF_MONTH, -7);
query.setAsOf(c.getTime());

QueryResult result = services.retrieve(query);

for (Asset story : result.getAssets()) {
    System.out.println(story.getId().getToken());
    System.out.println(story.getAttribute(nameAttribute).getValue());
    System.out.println(story.getAttribute(estimateAttribute).getValue());
    System.out.println();
}

/***** OUTPUT *****/
Story:1063
Logon
3

Story:1064
Add Customer Details
1

Story:1065
Add Customer Header
3
*****/
```





Querying Configurations and Localizations

In addition to working with VersionOne [assets](#), the Java SDK provides read-only access to a subset of system configurations and localizations to allow for client-side data validation.

For system configurations, settings for Effort Tracking, Story Tracking Level, Defect Tracking Level are available so that entry of Effort, Detail Estimate, and ToDo can be done consistently with the way VersionOne is configured.

For system localizations, you can look up the value used within the VersionOne user interface based on the asset's or attribute's system name.

Prior to querying configurations and localizations in VersionOne, you must first instantiate a **V1Connector** object:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKlk+PnyFkEWVX5KI2u6Jk=")
    .build();

IServices services = new Services(connector);
```

In versions of the SDK prior to the 15.0.0.0 release, you would also have to instantiate a connector for the **MetaModel** object. However, starting with the 15.0.0.0 release, that is no longer necessary.

The **MetaModel** object is now available from the **getMeta** method of the **Services** object. For more advanced **Services** constructor overloads, see the [Creating a Services Object](#) topic.



Querying System Configurations

While working with VersionOne assets requires the use of the **Services** object, accessing the system configurations requires using the **V1Configuration** object. This example shows how to get the available system settings using the **V1Configuration** object:

```
V1Configuration configuration = new V1Configuration(connector);

System.out.println(String.format("Effort tracking level: %s", configuration.isEffortTracking()));
System.out.println(String.format("Story tracking level: %s", configuration.getStoryTrackingLevel()));
System.out.println(String.format("Defect tracking level: %s", configuration.getDefectTrackingLevel()));
System.out.println(String.format("Capacity planning: %s", configuration.getCapacityPlanning()));
System.out.println(String.format("Maximum attachment size: %s", configuration.getMaxAttachmentSize()));

/***** OUTPUT *****/
Effort tracking level: True
Story tracking level: Mix
Defect tracking level: Mix
Capacity planning: ByMemberByTeam
Maximum attachment size: 4194304
*****/
```

Detail Estimate, ToDo and Effort can be entered for Stories and Defects, or for their child Tasks and Tests, depending on how the system is configured. The **StoryTrackingLevel** and **DefectTrackingLevel** properties indicate where input of Detail Estimate, ToDo and Effort are taken.

A value of "True" indicates that Detail Estimate, ToDo, and Effort input is accepted at the PrimaryWorkitem level only. A value of "False" indicates that Detail Estimate, ToDo, and Effort input is accepted at the Task/Test level only. A value of "Mix" indicates that Detail Estimate, ToDo, and Effort input is accepted at both the PrimaryWorkitem and Task/Test level.

Querying System Localizations

Accessing system localizations is accomplished via the **Services** object using its **getLocalization** method, and there are three approaches that it supports.

The first approach is used for getting the localized name of an asset based on its system name:

```
System.out.println(String.format("Timebox name: %s", services.getLocalization("Timebox")));
System.out.println(String.format("Scope name: %s", services.getLocalization("Scope")));
System.out.println(String.format("Epic name: %s", services.getLocalization("Epic")));
System.out.println(String.format("Story name: %s", services.getLocalization("Story")));
System.out.println(String.format("Defect name: %s", services.getLocalization("Defect")));

/***** OUTPUT *****/
Timebox: Iteration
Scope: Project
Epic: Portfolio Item
```



Story: Story
Defect: Defect
*****/

The second approach is used for getting the localized value of a single attribute based on its attribute definition:

```
IAttributeDefinition scopeNameAttribute = services.getAttributeDefinition("Scope.Name");
IAttributeDefinition timeboxNameAttribute = services.getAttributeDefinition("Timebox.Name");

System.out.println(String.format("Scope name attribute: %s", services.getLocalization(timeboxNameAttribute)));
System.out.println(String.format("Timebox name attribute: %s", services.getLocalization(scopeNameAttribute)));

/**** OUTPUT ****
Scope name attribute: Title
Timebox name attribute: Title
*****/
```

The third approach is used for getting the localized values of multiple attributes based on their attribute definitions:

```
IAttributeDefinition nameAttribute = services.getAttributeDefinition("Story.Name");
IAttributeDefinition estimateAttribute = services.getAttributeDefinition("Story.Estimate");

ArrayList<IAttributeDefinition> attributes = new ArrayList<IAttributeDefinition>(Arrays.asList(nameAttribute,
estimateAttribute));
Map<String, String> localizations = services.getLocalization(attributes);

System.out.println(String.format("Story name attribute: %s", localizations.get(nameAttribute.getToken())));
System.out.println(String.format("Story estimate attribute: %s", localizations.get(estimateAttribute.getToken())));

/**** OUTPUT ****
Story name attribute: Title
Story estimate attribute: Estimate Pts.
*****/
```



Updating Assets

Updating [assets](#) through the Java SDK involves calling the Save method of the Services object. The process is that you first have to query for the asset to update, make the update in memory, then save the asset back to VersionOne.

Prior to updating an asset in VersionOne, you must first instantiate a **V1Connector** and **Services** object:

```
V1Connector connector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKLk+PnyFxEWVX5KI2u6Jk=")
    .build();

IServices services = new Services(connector);
```

In versions of the SDK prior to the 15.0.0.0 release, you would also have to instantiate a connector for the **MetaModel** object. However, starting with the 15.0.0.0 release, that is no longer necessary.

The **MetaModel** object is now available from the **getMeta** method of the **Services** object. For more advanced **Services** constructor overloads, see the [Creating a Services Object](#) topic.

Updating a Scalar Value Attribute

This example shows that updating a scalar [attribute](#) on an asset is accomplished by calling the **setAttributeValue** method on an asset, specifying the attribute definition of the attribute you wish to change and the new scalar value. This example updates the Name attribute on the Story with ID 7617:



```

Oid storyId = services.getOid("Story:7617");

Query query = new Query(storyId);
IAssetType storyType = services.getMeta().getAssetType("Story");
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
query.getSelection().add(nameAttribute);
QueryResult result = services.retrieve(query);
Asset story = result.getAssets()[0];
String oldName = story.getAttribute(nameAttribute).getValue().toString();
story.setAttributeValue(nameAttribute, "New Name");
services.save(story);

System.out.println(story.getOid().getToken());
System.out.println(oldName);
System.out.println(story.getAttribute(nameAttribute).getValue());

/***** OUTPUT *****/
Story:7617:9244
My New Story
New Name
*****/

```

Updating a Single-Value Relation Attribute

This example shows that updating a single-value relation is accomplished by calling the **setAttributeValue** method on an asset, specifying the attribute definition of the attribute you wish to change and the ID for the new relation. This example updates the source of the Story with ID 7617:

```

Oid storyId = services.getOid("Story:7617");

Query query = new Query(storyId);
IAssetType storyType = services.getMeta().getAssetType("Story");
IAttributeDefinition sourceAttribute = storyType.getAttributeDefinition("Source");
query.getSelection().add(sourceAttribute);
QueryResult result = services.retrieve(query);
Asset story = result.getAssets()[0];
String oldSource = story.getAttribute(sourceAttribute).getValue().toString();
story.setAttributeValue(sourceAttribute, "StorySource:149");
services.save(story);

System.out.println(story.getOid().getToken());
System.out.println(oldSource);
System.out.println(story.getAttribute(sourceAttribute).getValue());

/***** OUTPUT *****/
Story:7617:9245
NULL
StorySource:149
*****/

```



Updating a Multi-Value Relation Attribute

This example shows that updating a multi-value relation is accomplished by calling either the **removeAttributeValue** or **addAttributeValue** methods on an asset, specifying the attribute definition of the attribute you wish to change and the ID of the relation you wish to add or remove. This example updates one Member and removes another Member from the Story with ID 7617:

```
Oid storyId = services.getOid("Story:7617");

Query query = new Query(storyId);
IAssetType storyType = services.getMeta().getAssetType("Story");
IAttributeDefinition ownersAttribute = storyType.getAttributeDefinition("Owners");
query.getSelection().add(ownersAttribute);
QueryResult result = services.retrieve(query);
Asset story = result.getAssets()[0];

List<Object> oldOwners = new ArrayList<Object>();
oldOwners.addAll(Arrays.asList(story.getAttribute(ownersAttribute).getValues()));
story.removeAttributeValue(ownersAttribute, "Member:20");
story.addAttributeValue(ownersAttribute, "Member:2024");
services.save(story);
System.out.println(story.getOid().getToken());
Iterator<Object> iter = oldOwners.iterator();

while (iter.hasNext()) {
    Oid oid = (Oid) iter.next();
    System.out.println(oid.getToken());
}

for (Object o : story.getAttribute(ownersAttribute).getValues()) {
    Oid oid = (Oid) o;
    System.out.println(oid.getToken());
}

/***** OUTPUT *****/
Story:7617:9247
Member:20
Member:2024
*****/
```



Working with Attachments and Images

Working with attachments and images via the Java SDK is slightly different from other types of assets in that it involves working with specialized endpoints. These specialized endpoints are used for uploading and downloading the attachment and image data, which in most cases is in a binary format. Adding attachment and images to a VersionOne asset is a two step process:

Creating an Attachment

Prior to the 15.1.0 release of the SDK, creating an attachment for an asset involved first creating an Attachment asset then using the **Attachments** object to upload the attachment data. Starting with the 15.1.0 release, a much simpler **saveAttachment** method is available on the **Services** object. With the **saveAttachment** method, you no longer have to create a special **V1Connector** to the attachment.img endpoint to upload attachments to VersionOne.

Creating an Attachment Using the Services Object

To create an attachment and associate it with an asset in VersionOne, call the **saveAttachment** method of the **Services** object passing in the path and name of the file, the asset to associate with, and the name to use for the attachment. This example first creates a new story then adds the attachment to it:

```
Oid projectId = services.getOid("Scope:0");
IAssetType storyType = services.getMeta().getAssetType("Story");
Asset newStory = services.createNew(storyType, projectId);
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
newStory.setAttributeValue(nameAttribute, "Story with Attachment");
services.save(newStory);
```



```
String file = "C:\\Temp\\versionone.jpg";
services.saveAttachment(file, newStory, "Attachment for " + newStory.getOid().toString());
```

Creating an Attachment Using the Attachment Object

To create an attachment and associate it with an asset in VersionOne, you must first create an Attachment asset and retain its [OID](#) value which will be used to upload the attachment. This example creates an attachment asset for the Story with ID 1317.

```
Oid storyOid = services.getOid("Story:1317");
String file = "C:\\Temp\\versionone.png";
String mimeType = MimeTypes.resolve(file);

IAssetType attachmentType = services.getMeta().getAssetType("Attachment");
IAttributeDefinition attachmentAssetDef = attachmentType.getAttributeDefinition("Asset");
IAttributeDefinition attachmentContent = attachmentType.getAttributeDefinition("Content");
IAttributeDefinition attachmentContentType = attachmentType.getAttributeDefinition("ContentType");
IAttributeDefinition attachmentFileName = attachmentType.getAttributeDefinition("Filename");
IAttributeDefinition attachmentName = attachmentType.getAttributeDefinition("Name");
Asset attachment = services.createNew(attachmentType, Oid.Null);
attachment.setAttributeValue(attachmentName, "Attachment for " + storyOid.getMomentless());
attachment.setAttributeValue(attachmentFileName, file);
attachment.setAttributeValue(attachmentContentType, mimeType);
attachment.setAttributeValue(attachmentContent, "");
attachment.setAttributeValue(attachmentAssetDef, storyOid);
services.save(attachment);

Oid attachmentOid = attachment.getOid();
```

Note the use of the **MimeTypes** helper object. The **resolve** method of this object is used to determine the content type to use for the attachment asset. Also note that an empty string is passed for the Content attribute.

Once you've created the attachment asset and have its OID, you can then use the **Attachments** object to upload the binary data of the attachment. To use the **Attachments** object, you'll need to create a new **V1Connector** object, using the **useEndpoint** method to set the attachment.img endpoint. You then use the **java.io** package to read the binary data of the attachment, and then methods of the attachment class to write the stream.

```
V1Connector attachmentConnector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKlk+PnyFxEWVX5Kl2u6Jk=")
    .useEndpoint("attachment.img/")
    .build();

Attachments attachments = new Attachments(attachmentConnector);

FileInputStream inStream = new FileInputStream(file);
OutputStream output = attachments.getWriter(attachmentOid.getKey().toString(), mimeType);
```



```

byte[] buffer = new byte[inStream.available() + 1];
while (true) {
    int read = inStream.read(buffer, 0, buffer.length);
    if (read <= 0)
        break;
    output.write(buffer, 0, read);
}

attachments.setWriter(attachmentOid.getKey().toString());
inStream.close();

```

Note that as of the 15.1.0 release of the SDK, both the **useEndpoint** method and the **Attachments** object have been marked for deprecation.

Querying an Attachment

Prior to the 15.1.0 release of the SDK, querying an attachment for an asset involved using the **getReadStream** method of the **Attachments** object to download the attachment data. Starting with the 15.1.0.0 release, a much simpler **getAttachment** method is available on the **Services** object. With the **getAttachment** method, you no longer have to create a special **V1Connector** to the attachment.img endpoint to download attachments from VersionOne.

Querying an Attachment Using the Services Object

Getting an attachment from VersionOne involves getting the OID of the attachment, then using the **getAttachment** method of the **Services** object. In this example, a query is used to get all the attachments associated with Story ID 6052, and then write each attachment as a file:

```

Oid assetOid = services.getOid("Story:1181");

IAssetType attachmentType = services.getMeta().getAssetType("Attachment");
Query query = new Query(attachmentType);
IAttributeDefinition filenameAttribute = attachmentType.getAttributeDefinition("Filename");
IAttributeDefinition assetAttribute = attachmentType.getAttributeDefinition("Asset");
query.getSelection().add(filenameAttribute);
query.getSelection().add(assetAttribute);

FilterTerm term = new FilterTerm(assetAttribute);
term.equal(assetOid.getMomentless());
query.setFilter(term);
QueryResult result = services.retrieve(query);

for (Asset attachment : result.getAssets()) {

    String fileName = attachment.getAttribute(filenameAttribute).getValue().toString();
    File file = new File(fileName);
    InputStream inStream = _services.getAttachment(attachment.getOid());

    if (null != inStream) {

        FileOutputStream outStream = new FileOutputStream(file);

```



```

byte[] buf = new byte[1024];
int len;

while ((len = inStream.read(buf)) > 0) {
    try {
        outStream.write(buf, 0, len);
    } catch (IndexOutOfBoundsException e) {
        System.out.println("\nIndexOutOfBoundsException occurred, please try again.....\n");
    }
}

outStream.close();
inStream.close();
}

System.out.println(fileName);
}

/**** OUTPUT ****
Test Image Attachment.png
Test Document Attachment.pdf
*****/

```

Querying an Attachment Using the Attachment Object

Getting an attachment from VersionOne involves getting the OID of the attachment, then using the **getReader** method of the **Attachments** object. In this example, a query is used to get all the attachments associated with Story ID 1317, and then write the attachments as files to the C:\Temp directory:

```

Oid assetOid = services.getOid("Story:1317");

V1Connector attachmentConnector = V1Connector
    .withInstanceUri("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKlk+PnyFxxEWVX5KI2u6Jk=")
    .useEndpoint("attachment.img/")
    .build();

Attachments attachments = new Attachments(attachmentConnector);

IAssetType attachmentType = services.getMeta().getAssetType("Attachment");
Query query = new Query(attachmentType);
IAttributeDefinition filenameAttribute = attachmentType.getAttributeDefinition("Filename");
IAttributeDefinition assetAttribute = attachmentType.getAttributeDefinition("Asset");
query.getSelection().add(filenameAttribute);
query.getSelection().add(assetAttribute);

FilterTerm term = new FilterTerm(assetAttribute);
term.equal(assetOid.getMomentless());
query.setFilter(term);
QueryResult result = services.retrieve(query);

```



```

String filePath = "C:\\Temp\\";
for (Asset attachment : result.getAssets()) {

    String fileName = attachment.getAttribute(filenameAttribute).getValue().toString();
    String attachmentKey = attachment.getOid().getKey().toString();
    File file = new File(filePath + "v1_" + fileName);

    InputStream inStream = attachments.getReader(attachmentKey);

    if (null != inStream) {
        OutputStream outStream = new FileOutputStream(file);
        byte buf[] = new byte[1024];
        int len;
        while ((len = inStream.read(buf)) > 0) {
            outStream.write(buf, 0, len);
        }
        outStream.close();
        inStream.close();
    }
    System.out.println(fileName);
}

/***** OUTPUT *****/
Test Image Attachment.png
Test Document Attachment.pdf
*****/

```

Note that as of the 15.1.0 release of the SDK, both the **useEndpoint** method and the **Attachments** object have been marked for deprecation.

Deleting an Attachment

Deleting an attachment is the same process as used for any other VersionOne asset, you use the **getOperation** method to get the operation to execute, then call the **executeOperation** method of the **Services** object, passing in the operation and the OID of the asset:

```

IOperation deleteOperation = services.getMeta().getOperation("Attachment.Delete");
services.executeOperation(deleteOperation, services.getOid("Attachment:6640"));

```

Adding an Embedded Image

Prior to the 15.1.0 release of the SDK, creating an embedded image for an asset involved first creating an **EmbeddedImage** asset then using the **Attachments** object to upload the embedded image data. Starting with the 15.1.0 release, a much simpler **saveEmbeddedImage** method is available on the **Services** object. With the **saveEmbeddedImage** method, you no longer have to create a special **V1Connector** to the embedded.img endpoint to upload embedded images to VersionOne.



Adding an Embedded Image Using the Services Object

Adding an embedded image using the **Services** object involves calling the **SaveEmbeddedImage** method and specifying the path and name to the image to embed, and the asset that it should be associated with.

In this example, a new story is created, an embedded image is associated with it, then the story's description field is modified to include the embedded image:

```
//Create a new story.
IAssetType storyType = services.getMeta().getAssetType("Story");
Asset newStory = services.createNew(storyType, services.getOid("Scope:0"));
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
IAttributeDefinition descriptionAttribute = storyType.getAttributeDefinition("Description");
String name = "Story with an embedded image";
newStory.setAttributeValue(nameAttribute, name);
services.save(newStory);

//Add the embedded image to the story.
String file = "C:\\Temp\\versionone.jpg";
Oid embeddedImageOid = services.saveEmbeddedImage(file, newStory);
String embeddedImageTag = "<img src=\"" + embeddedImageOid.getKey() + " alt=\"\" data-oid=\"" + embeddedImageOid.getMomentless() + "\" />";
newStory.setAttributeValue(descriptionAttribute, embeddedImageTag);
services.save(newStory);
```

Adding an Embedded Image Using the Attachments Object

An embedded image is an image that you add to the Description attribute of an asset, and adding an embedded image follows a similar process as that of an attachment, except that it makes use of the embedded.img endpoint, and involves using the EmbeddedImage asset and adding a bit of HTML.

In the following example, a story is created in Scope:0 and an image file is read from the C:\Temp directory and added as an embedded image to the description of the story:

```
//Create a new story.
IAssetType storyType = services.getMeta().getAssetType("Story");
Asset newStory = services.createNew(storyType, services.getOid("Scope:0"));
IAttributeDefinition nameAttribute = storyType.getAttributeDefinition("Name");
IAttributeDefinition descriptionAttribute = storyType.getAttributeDefinition("Description");
String name = "Story with an embedded image";
newStory.setAttributeValue(nameAttribute, name);
services.save(newStory);

//Create an embedded image asset.
String fileName = "versionone.png";
String filePath = "C:\\Temp\\";
String mimeType = MimeType.resolve(fileName);

IAssetType embeddedImageType = services.getMeta().getAssetType("EmbeddedImage");
```



```

Asset newEmbeddedImage = services.createNew(embeddedImageType, ObjectId.Null);
IAttributeDefinition assetAttribute = embeddedImageType.getAttributeDefinition("Asset");
IAttributeDefinition contentTypeAttribute = embeddedImageType.getAttributeDefinition("ContentType");
IAttributeDefinition contentTypeAttribute = embeddedImageType.getAttributeDefinition("ContentType");
newEmbeddedImage.setAttributeValue(assetAttribute, newStory.getId());
newEmbeddedImage.setAttributeValue(contentTypeAttribute, mimeType);
newEmbeddedImage.setAttributeValue(contentAttribute, "");
services.save(newEmbeddedImage);
String key = newEmbeddedImage.getId().getKey().toString();

//Save the embedded image file data.
V1Connector attachmentConnector = V1Connector
    .withInstanceUrl("<Server Base URI>")
    .withUserAgentHeader("AppName", "1.0")
    .withAccessToken("1.rWM8lKLk+PnyFxEWVX5Kl2u6Jk=")
    .useEndpoint("embedded.img/")
    .build();

IAttachments attachments = new Attachments(attachmentConnector);

FileInputStream inStream = new FileInputStream(filePath + fileName);
OutputStream output = attachments.getWriter(key, mimeType);
byte[] buffer = new byte[inStream.available() + 1];
while (true) {
    int read = inStream.read(buffer, 0, buffer.length);
    if (read <= 0)
        break;
    output.write(buffer, 0, read);
}

attachments.getWriter(key);
inStream.close();

//Add the embedded image to the story.
newStory.setAttributeValue(descriptionAttribute, "<img src=\"" + embedded.img/" + key + " alt=\"\" data-oid=" +
newEmbeddedImage.getId().getMomentless() + " />");
services.save(newStory);

```

Note that as of the 15.1.0.0 release of the SDK, both the **useEndpoint** method and the **Attachments** object have been marked for deprecation.

