

# HowTo use Mercury for accessing repositories

Mercury provides an implementation-neutral way to access GAV-based repositories, including AV repositories, like OSGi. OSGi access is not implemented yet. By access I mean reading artifacts and metadata from repositories and writing artifacts to repositories, metadata is updated by writes.

- [APIs](#)
- [Build Details](#)

## APIs

Upper level API works like the following:

- create a bunch of **Repository** objects
- obtain an instance of **DependencyProcessor** implementation
  - if you don't plan to use **getDependencies** call, you can use **DependencyProcessor.NULL\_PROCESSOR**
- create an instance of **VirtualRepositoryReader** by passing the repositories and dependency processor
- use it

All the calls accept a collection of requests as an input and return an object that hides **getResults**, that normally is a `map< queryElement, Collection<ResultObject> >` response. The response object has convenience methods **hasExceptions()**, **hasResults()**, **getExceptions()**, **getResults()**

One of the key building blocks is a hierarchy of Artifact data:

- **ArtifactCoordinates** - is truly the 3 components GAV
- **ArtifactBasicMetadata** - is coordinates plus type/classifier plus convenience methods like hash calculation and such
- **ArtifactMetadata** adds a list of dependency objects, captured as **ArtifactBasicMetadata**
- **DefaultArtifact** implements **Artifact** interface and adds `pomBlob (byte[])` and `file`, that points to actual binary

Let's assume that you obtained virtual reader in a variable, called **vr**. The "use it" part means the following:

- **vr.readVersions( Collection<ArtifactBasicMetadata> query )** this is **the only** call that interprets queries. All others treat supplied versions as exact, not as queries.
  - this one takes a collection of queries in the form of **ArtifactBasicMetadata** and returns a collection of good old **ArtifactBasicMetadata** objects. The input object can have a query in the version field
  - version query is now accepted as:
    - 1.2.3 - means any version, higher or equal to 1.2.3, variations:
      - 1.2.3-SNAPSHOT - find latest snapshot of this version
      - 1.2.3-LATEST - find latest snapshot or release of this version
      - 1.2.3-RELEASE - find latest release of this version
    - range in the form of
      - [1.2.3,) this is equal to simple 1.2.3 query
      - (1.2.3,) same as previous, but 1.2.3 version is excluded from results
      - all other obvious variations of `/(V1,V2)`
- **vr.readArtifacts(Collection<ArtifactBasicMetadata> query)** - retrieve full-blown **Artifact** for each query element. Artifacts are charged with `pomBytes()` and a reference to a local file
- **vr.readDependencies( Collection<ArtifactBasicMetadata> query )** - returns a `List<ArtifactBasicMetadata>` of dependencies for each element of the query. In other words - it returns an instance of **ArtifactMetadata** for each **ArtifactBasicMetadata** from query. This call is mostly for dependency tree builder. And of course - for you, if you find a creative way to utilize it 😊

## Build Details

To write client code that can read and write repositories, declare the following dependencies:

```
<dependency>

<groupId>org.apache.maven.mercury</groupId>

<artifactId>mercury-repo-local-m2</artifactId>

    <version>${mercury.version}</version>
</dependency>
<dependency>

<groupId>org.apache.maven.mercury</groupId>

<artifactId>mercury-repo-remote-m2</artifactId>

    <version>${mercury.version}</version>
</dependency>
<dependency>

<groupId>org.apache.maven.mercury</groupId>

<artifactId>mercury-repo-virtual</artifactId>

    <version>${mercury.version}</version>
</dependency>
```

Then assemble an instance of **VirtualRepositoryReader** for reading from multiple repos, or create a Repository object, obtain **RepositoryWriter** from it and deploy stuff to that repository for you heart's pleasure.

Sample code from **VirtualRepositoryReaderTest**

```
File                _testBase;
LocalRepository    _localRepo;
Server             _server;
RemoteRepository   _remoteRepo;
VirtualRepositoryReader _vr;

    _testBase = new File( "/my/local/repo"
);

    _localRepo = new LocalRepositoryM2(
"localRepo", _testBase );

    _server = new Server( "remoteRepo", new
URL("http://repo1.maven.org/maven2") );

    _remoteRepo = new RemoteRepositoryM2(
_server.getId(), _server );

    List<Repository> rl = new
ArrayList<Repository>();
    rl.add( _localRepo );
    rl.add( _remoteRepo );

    // null dependency processor as I don't
use readDependencies()
    _vr = new VirtualRepositoryReader( rl,
DependencyProcessor.NULL_PROCESSOR );
```