

# Oracle Enterprise Repository

## Extraction API

### Overview

As part of the **Use - Download** (extraction) process the user is asked to associate the selected asset with a particular project. These instances of usage, along with surveys (which are included in usage updates) are the primary drivers for metrics within Oracle Enterprise Repository.

---

- **Note on the use of the term "extraction."**
    - In earlier product releases the term *Extraction* was used to describe the act of downloading or otherwise accessing an asset's payload. The term *Extraction* has since been replaced by the phrase **Use - Download**. Please note, however, that within the context of this **Extraction API** document, most instances of use of the term *Extraction* (particularly in code examples) were left intact in order to simplify the use of REX API.
- 

### Definitions

- **State** *State* refers to the usage status of an asset that has been selected for use/download. There are four possible states:
  - IN PROCESS
  - ACCEPTED
  - REJECTED
  - DEPLOYED (DEPLOYED is covered under Projects).
- **Extraction Download** Contains the file info associated with the extracted asset. Values for an extraction download can be 0 or 1.
- **File Info** Information and URL links to the actual files associated with the asset make up the File Info as contained in an extraction download. File Info values for an extraction download can be 0 to n.
- **Related Asset** Within Oracle Enterprise Repository, a given asset can be associated with others through a number of pre-defined and/or custom-configured relationships. An asset can contain 0 to n related assets.

### Related Subsystems

- [AssetSubsystem](#)
- [ProjectSubsystem](#)
- [CategorizationTypeSubsystem](#)
- [SurveySubsystem](#)

## Use Cases

### Use Case: Extract an Asset

#### *Description*

An Extraction is created when an asset is associated for use in a project by the user. A list of related assets is also made available for extraction during this process. In this case the user can simultaneously extract both the primary asset and any related assets. A unique extraction is then recorded for each asset. Creating an extraction results in an array containing 0 to n extraction downloads. The file info value for each download can be 0 to n. The file info contains information about the file. This information can be used to create a link to the file.

In order to extract an asset the following conditions must be met:

- The user must be a member of the project to which the asset is to be extracted.
- The user must be assigned the appropriate role type(s).
- The project must be open.
- The asset(s) must be registered and active.
- If Custom Access Settings are enabled, the user performing the extraction must have appropriate access rights to the specified asset(s).
- If Custom Access Settings are enabled, the user performing the extraction will receive file info only for those files to which the user has the appropriate permissions.
- These conditions are checked by the appropriate methods in which they are used. Exceptions are thrown if the conditions are not met.

#### Sample Code:

```
package com.flashline.sample.extractionapi;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import javax.xml.rpc.ServiceException;

import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
```

```

import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Extraction;
import com.flashline.registry.openapi.entity.ExtractionDownload;
import com.flashline.registry.openapi.entity.FileInfo;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;

public class ExtractAsset {
    public static void main(String pArgs[]) throws OpenAPIException, RemoteException,
        ServiceException {
    try {
        ////////////////////////////////////////////////////
        // Connect to Oracle Enterprise Repository
        ////////////////////////////////////////////////////
        URL IURL = null;
        IURL = new URL(pArgs[0]);
        FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
            .getFlashlineRegistry(IURL);

        ////////////////////////////////////////////////////
        // Authenticate with OER
        ////////////////////////////////////////////////////
        AuthToken authToken = repository.authTokenCreate(
            pArgs[1], pArgs[2]);

        long ASSET_ID_1 = 589;    // must be a valid asset id in OER
        long ASSET_ID_2 = 569;    // must be a valid asset id in OER
        long PROJECT_ID = 50000; // must be a valid project id in OER
        long EXTRACTION_ID = 0;

        // -----
        // Create a new extraction
        long[] lAssetIDs = { ASSET_ID_1, ASSET_ID_2 };
        ExtractionDownload[] extractionDownloads = repository.extractionCreate(authToken, PROJECT_ID, lAssetIDs);
        System.out.println("Number of new extraction downloads created: " + extractionDownloads.length);

        // -----
        // Read an extraction by project and asset
        Extraction extraction = repository.extractionReadByProjectAndAsset(authToken, PROJECT_ID, ASSET_ID_1);
        EXTRACTION_ID = extraction.getID();

        // -----
        // Read an extraction by ID
        Extraction extractionByID = repository.extractionRead(authToken, EXTRACTION_ID);
        System.out.println("The extraction '"+extractionByID.getDisplayName()+"' was read by id ('+EXTRACTION_ID+");

        // -----
        // Read asset extractions
        Extraction[] assetExtractions = repository.extractionReadAssetExtractions(authToken, PROJECT_ID, ASSET_ID_1, true);
        System.out.println("The number of extractions for this asset is: "+(assetExtractions==null ? 0 : assetExtractions.length));

        // -----
        // Read project extractions
        Extraction[] projectExtractions = repository.extractionReadProjectExtractions(authToken, PROJECT_ID, true);
        System.out.println("The number of extractions for this project is: "+(projectExtractions==null ? 0 : projectExtractions.length));
    }
}

```

```

// -----
// Read related assets
Asset[] relatedAssets = repository.extractionReadRelatedAssets(authToken, ASSET_ID_2);
System.out.println("The number of related assets is: "+relatedAssets==null ? 0 : relatedAssets.length);

// -----
// Read File-Info for an extraction
List fileInfosList = new ArrayList();
if (projectExtractions != null) {
    for (int i = 0; i < projectExtractions.length; i++) {
        extraction = repository.extractionRead(authToken, projectExtractions[i].getID());
        fileInfosList.add(repository.extractionReadFileInfos(authToken, extraction));
    }
}

// -----
// Get File
List fileInfoList = new ArrayList();
Iterator fileInfoListIter = fileInfosList.iterator();
while (fileInfoListIter.hasNext()) {
    FileInfo[] fileInfos = (FileInfo[]) fileInfoListIter.next();
    for (int i = 0; i < fileInfos.length; i++) {
        fileInfoList.add(fileInfos[i]);
    }
}
String[] fileLinks = new String[fileInfoList.size()];
for (int i = 0; i < fileInfoList.size(); i++) {
    FileInfo fileInfo = (FileInfo) fileInfoList.get(i);
    fileLinks[i] = repository.repositoryFileTranslator(authToken, fileInfo);
    System.out.println("Project extraction file-info link: "+fileLinks[i]);
}

// -----
// revert extractions
repository.extractionResetDatabase();

} catch (OpenAPIException IEx) {
    System.out.println("ServerCode = " + IEx.getServerErrorCode());
    System.out.println("Message = " + IEx.getMessage());
    System.out.println("StackTrace:");
    IEx.printStackTrace();
} catch (RemoteException IEx) {
    IEx.printStackTrace();
} catch (ServiceException IEx) {
    IEx.printStackTrace();
} catch (MalformedURLException IEx) {
    IEx.printStackTrace();
}
}
}

```

### *Notes about FileInfo Objects*

FileInfo objects represent individual files associated with the extracted asset. The physical location of the file may be obtained by the two following methods.

1. Using the downloadURI property on the FileInfo object itself, i.e. fileInfo.getDownloadURI().
  2. Using the OpenAPI method repositoryFileTranslator passing the FileInfo object, i.e. flashlineRegistry.repositoryFileTranslator(authToken, fileInfo).
- DO NOT use the URI property on the FileInfo object which represents an Oracle Enterprise Repository specific path.

## Use Case: Read an Extraction

### *Description*

Several methods beyond those covered in the Extract an Asset use case can be used to read extractions. Extractions can be grouped by asset, project, or user. The specific grouping of extractions determines the method to be used.

In order to read an extraction the following conditions must be met:

- The project must be open.
- The asset(s) must be registered and active.
- The extraction must be active.

These conditions are checked by the appropriate methods in which they are used. Exceptions are thrown if the conditions are not met.

### Sample Code:

```
package com.flashline.sample.extractionapi;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Extraction;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;

public class ExtractRead {
    public static void main(String pArgs[]) throws OpenAPIException, RemoteException,
        ServiceException {
        try {
            ////////////////////////////////////////////////////
            // Connect to Oracle Enterprise Repository
            ////////////////////////////////////////////////////
        }
    }
}
```

```

URL IURL = null;
IURL = new URL(pArgs[0]);
FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
    .getFlashlineRegistry(IURL);

// //////////////////////////////////////
// Authenticate with OER
// //////////////////////////////////////
AuthToken authToken = repository.authTokenCreate(
    pArgs[1],pArgs[2]);

long PROJECT_ID = 50000; // must be a valid project id in the OER
long ASSET_ID = 569; // must be a valid asset id in the OER

// -----
// Read project extractions
Extraction[] projectExtractions = repository
    .extractionReadProjectExtractions(authToken, PROJECT_ID, true);

// -----
// Read asset extractions
Extraction[] assetExtractions = repository
    .extractionReadAssetExtractions(authToken, PROJECT_ID, ASSET_ID, true);

// -----
// Read user extractions
Extraction[] userExtractions = repository
    .extractionReadUserExtractions(authToken, true);

// -----
// Read related assets
Asset[] assets = repository.extractionReadRelatedAssets(authToken,
    ASSET_ID);
} catch (OpenAPIException IEx) {
    System.out.println("ServerCode = " + IEx.getServerErrorCode());
    System.out.println("Message = " + IEx.getMessage());
    System.out.println("StackTrace:");
    IEx.printStackTrace();
} catch (RemoteException IEx) {
    IEx.printStackTrace();
} catch (ServiceException IEx) {
    IEx.printStackTrace();
} catch (MalformedURLException IEx) {
    IEx.printStackTrace();
}
}
}

```

## Use Case: Update an Extraction

### *Description*

An extraction record is updated when the state of the asset is changed or when the consumer of the asset completes an asset survey. A state change or survey completion can be separate transactions or performed in tandem.

In order to update an extraction the following conditions must be met:

- The project must be open
- The asset must be registered and active.
- The extraction must be active.
- The survey taken must be active.

These conditions are checked by the appropriate methods in which they are used. Exceptions are thrown if the conditions are not met.

Sample Code:

```
package com.flashline.sample.extractionapi;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Answer;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Categorization;
import com.flashline.registry.openapi.entity.Choice;
import com.flashline.registry.openapi.entity.ChoiceList;
import com.flashline.registry.openapi.entity.Extraction;
import com.flashline.registry.openapi.entity.ExtractionDownload;
import com.flashline.registry.openapi.entity.IExtraction;
import com.flashline.registry.openapi.entity.Question;
import com.flashline.registry.openapi.entity.SurveyTaken;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;

public class ExtractUpdate {
    public static void main(String pArgs[]) throws OpenAPIException, RemoteException,
        ServiceException {
        try {
            ////////////////////////////////////////////////////////////////////
            // Connect to Oracle Enterprise Repository
            ////////////////////////////////////////////////////////////////////
            URL IURL = null;
            IURL = new URL(pArgs[0]);
            FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
                .getFlashlineRegistry(IURL);

            ////////////////////////////////////////////////////////////////////
            // Authenticate with OER
            ////////////////////////////////////////////////////////////////////
        }
    }
}
```

```

AuthToken authToken = repository.authTokenCreate(
    pArgs[1],pArgs[2]);

long PROJECT_ID = 50000; // must be a valid project id in the OER
long ASSET_ID = 569; // must be a valid asset id in the OER

// -----
// Create a new extraction
long[] lAssetIDs = { ASSET_ID };
ExtractionDownload[] extractionDownloads = repository.extractionCreate(authToken, PROJECT_ID, lAssetIDs);

Extraction[] assetExtractions = repository
    .extractionReadAssetExtractions(authToken, PROJECT_ID, ASSET_ID, true);

////////////////////////////////////
// this assumes that there is at least 1 extraction and the first one will
// be used
////////////////////////////////////
IExtraction iExtraction = repository
    .extractionReadExtractionStates(authToken);
Extraction extraction = repository.extractionRead(authToken,
    assetExtractions[0].getID());

////////////////////////////////////
// can set the status of the extraction to 'Deployed', 'Rejected', or 'In
// Process'.
////////////////////////////////////
assetExtractions[0].setStatus("In Process");

extraction = repository.extractionTentativelyAccept(authToken,
    extraction);
SurveyTaken surveyTaken = repository.surveyTakenRead(authToken,
    extraction);
extraction = repository.extractionUpdateSurvey(authToken,
    extraction, surveyTaken);
extraction.setStatus(iExtraction.getInProcess());

surveyTaken = repository.surveyTakenRead(authToken, extraction);
extraction = repository.extractionUpdateSurvey(authToken,
    extraction, surveyTaken);
Categorization[] rejectionReasons = repository
    .extractionReadRejectionReasons(authToken);

surveyTaken = repository.surveyTakenRead(authToken, extraction);
extraction = repository.extractionUpdateSurvey(authToken,
    extraction, surveyTaken);
surveyTaken = repository.surveyTakenRead(authToken, extraction);
Question[] questions = repository.surveyReadQuestions(authToken);

ChoiceList choiceList = null;
Choice[] choices = null;
Answer[] answers = new Answer[4];
for (int i = 0; i < answers.length; i++) {
    answers[i] = new Answer();
}

```

```

answers[0].setQuestionId(questions[0].getId());
choiceList = questions[0].getChoiceList();
choices = choiceList.getChoices();
answers[0].setChoiceId(choices[0].getId());
answers[0].setValue(choices[0].getValue());

answers[1].setQuestionId(questions[1].getId());
answers[1].setChoiceId(0);
answers[1].setValue("100");

answers[2].setQuestionId(questions[2].getId());
answers[2].setChoiceId(0);
answers[2].setValue("200");

answers[3].setQuestionId(questions[3].getId());
choiceList = questions[3].getChoiceList();
choices = choiceList.getChoices();
answers[3].setChoiceId(choices[3].getId());
answers[3].setValue(choices[3].getValue());

surveyTaken.setAnswers(answers);

surveyTaken = repository.surveyTakenUpdate(authToken, surveyTaken);

// -----
// revert extractions
repository.extractionResetDatabase();

} catch (OpenAPIException IEx) {
    System.out.println("ServerCode = " + IEx.getServerErrorCode());
    System.out.println("Message = " + IEx.getMessage());
    System.out.println("StackTrace:");
    IEx.printStackTrace();
} catch (RemoteException IEx) {
    IEx.printStackTrace();
} catch (ServiceException IEx) {
    IEx.printStackTrace();
} catch (MalformedURLException IEx) {
    IEx.printStackTrace();
}
}
}
}

```

### *Pitfalls*

Bear in mind that a state change or a survey taken is a two-step process. The first step is to change the state or take the survey. The second step is to update the extraction status using the `extractionUpdateStatus` method. A state change or survey taken can be separate transactions or performed in tandem. If performed in tandem only a single `extractionUpdateStatus` method call is required.

The `extractionUpdateStatus` method requires a `SurveyTaken`. This is true regardless of whether a survey was taken at the time the `extractionUpdateStatus`

method is called (as in a state change, for example). This survey is retrieved using the `surveyTakenRead` method. If a survey has not been taken for this extraction one will be created by the `surveyTakenRead` method.

The current survey in Oracle Enterprise Repository consists of four questions. When a survey is taken an array is created storing answers for the four questions. Each answer must contain three pieces of information in order to be valid:

- The value (the user response to the question)
- The question ID
- The choice ID.

Questions 2 and 3 are single answer questions, so the choice ID here will always be set to 0. Questions 1 and 4, however, are multiple-choice. The multiple choices are retrieved using the `surveyReadChoiceList` method.

### *Methods to avoid*

The following objects are used in the Extraction and Survey subsystems:

- `Extraction`
- `ExtractionDownload`
- `FileInfo`
- `SurveyTaken`
- `Question`
- `ChoiceList`
- `Choice`
- `Answer`

The use of any of the get methods within these objects is acceptable. All the remaining methods - especially the set methods - should be avoided. The events provided by these remaining methods are covered by the methods in the Extraction and Survey subsystems.