Picturepark

Picturepark Asset Connector Framework

Information and samples about the Picturepark Asset Connector Framework, targeted at partners and integrators

Author: Stefan Seidl
Version: 1.0

Contents

Picturepark

1 Introduction

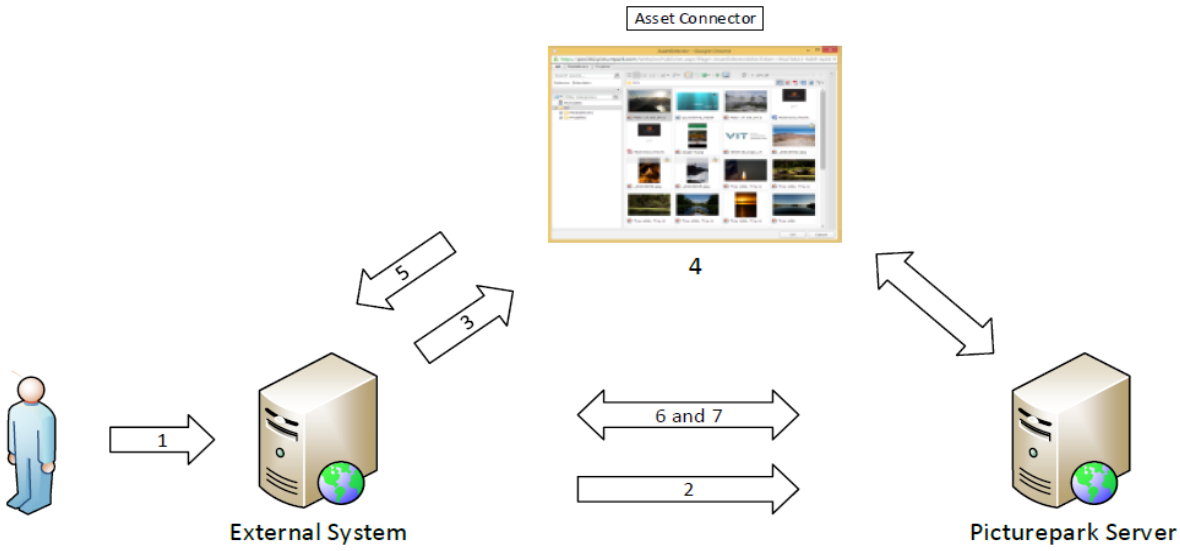
1.1 About Picturepark Asset Connector

Picturepark Asset Connector is a standardized framework used for integrations into 3rd party systems such as CMS, PIM or ERP systems that e.g. require a low or high resolution image or video to be embedded in a web page or a publication.

The PAC provides the well-known Picturepark UI so that users can easily browse, search and select the desired media assets from Picturepark such as images, PDF, movies or other media files stored in Picturepark. While the media asset is just a logical construct its output derivative (low-res jpg from a psd or h264 mpeg from a mov) is downloaded & referenced to the 3rd party system together with metadata, access permissions etc. that it carries.

1.2 Schematic process

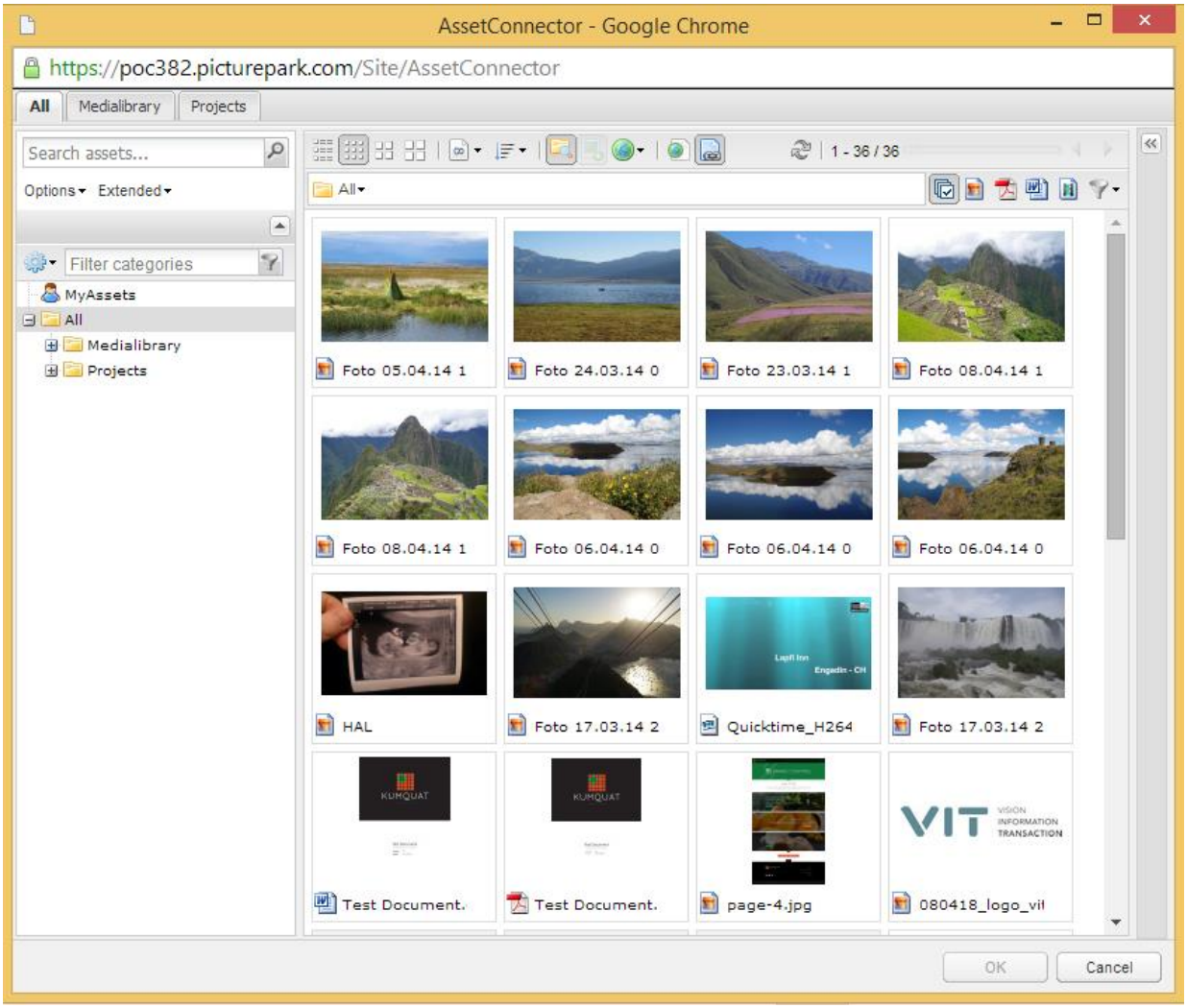
In the following scheme an Asset Connector process is described. A more technical description can be found in the next chapter. The 3rd party system is referenced as “external system”.



- The user initiates the process e.g. by clicking the “Place image here” button in the Content Management System (the external or calling system)
- The external system connects to the destination server and authenticates using a Security Token procedure to prevent the user from a manual logon. If this step is skipped, the user later is prompted to enter login/password information
- The external system calls the Asset Connector component that delivers the UI for asset selection submitting the Security Token for SSO or having the user login manually
- Using the Asset Connector component the user selects an asset and confirms his selection
- The Asset Connector component sends a JSON response containing the ID of the selected asset as well as the chosen derivative definition IDs to the redirect URL.
- The external system obtains the derivative IDs for each chosen derivative definition of the selected assets
- The external system retrieves the corresponding download path for the identified derivatives and then downloads or reference them for local storage and further use.

1.3 **Asset Connector Component**

The screenshot below shows the Asset Connector component. This component is based on Picturepark core code and can be adapted in many parts e.g. layout, design, asset filters, asset format filters etc.



2 Use Cases

2.1 Integration to a CMS System

The Content Manager who is working in the CMS wants to have access to the assets in Picturepark. He wants to place them on the Webpages or store them in a folder of the CMS for using them later. Another possibility is to link an asset from Picturepark a Webpage in the CMS. E.g. for streaming a video directly from Picturepark.

The following steps should be done for this integration:

1. Create a Webservice Session to Picturepark.
2. Call the Picturepark Asset Connector.
3. Wait until the user has selected one or more assets. The PAC will now send the download links to a redirect URL.
4. Optional: The user has the possibility to convert the asset, e.g. cropping or format conversion.
5. Download the assets or create an asset link, e.g. for the video streaming.
6. Optional: Get metadata for the downloaded assets from Picturepark.

2.2 Integration to a Product Information Management (PIM) System

The Product Manager creates new entries in the PIM System. He inputs texts and metadata to the entries. Then he wants to add product images to the entry. They are made by a photographer who has stored them in Picturepark. The Product Manager can now open Picturepark directly from the PIM System (via PAC) and select a product image for his entry. The image will automatically be downloaded in the correct format for the PIM system, and it will be assigned to the product / entry.

The following steps should be done for this integration:

1. Create a Webservice Session to Picturepark.
2. Call the Picturepark Asset Connector.
3. Wait until the user has selected one or more assets. The PAC will now send the download links to a redirect URL.
4. Optional: The user has the possibility to convert the asset, e.g. cropping or format conversion.
5. Download the correct derivative of the asset to the PIM system.
6. Optional: Get metadata for the downloaded assets from Picturepark.
7. Optional: Save metadata which is stored in the PIM system at the asset in Picturepark.

3 How to Implement

3.1 Before you start

Before you can start to integrate you need to ensure you're having the following information at hand:

- URL: Describing where you can find the test Picturepark instance.
- WSDL Basic: The public WSDL
- Customer ID: a number identifying the test Picturepark instance.
- GUID: A GUID needs to be used for developing against Picturepark's API
- User credentials: you will gain access to the system with a generic read user account
- Instance settings: you will get a summary of all important ID's related to your instance (derivative id's, language id's etc.)

Request on the website

3.2 Basic and extended Webservice

Please note that there are two different Web services: The public Webservice which provides read-only Webservice methods and is available for free with every Picturepark. Additionally, there is the extended Webservice which provides much more Webservice methods also for writing back to Picturepark but requires a license. Registered and certified developers and customers can integrate on both Web services.

3.3 Authentication - SOAP Webservice Proxy

You must first create a SOAP Webservice client (Webservice proxy). How to do this depends on your developing environment.

Within a DotNet environment the most convenient tool is VisualStudio. If it is not available, use the command line tool svcutil.exe.

See: http://Webservice8.picturepark.com/ExtendedPublicService.svc

Example in C# Code:

Step 1: Get a **clientGuid** from VIT or VIT partners, who will create the guid by registering the client with the server.

Step 2: Create a client (plus proxy-) server:

```
ExtendedPublicServiceClient PictureparkService = new ExtendedPublicServiceClient("Default");
```

If MTOM is available, use the MTOM binding in order to make streaming available for uploads (recommended):

```
ExtendedPublicServiceClient PictureparkService = new ExtendedPublicServiceClient("Mtom");
```

Step 3: You can now invoke the CreateSession werbservice, which will return a CoreInfo object, which contains the session informations and is needed for all other services:

```
CoreInfo coreInfo = PictureparkService.CreateSession(param1, param2, ..)
```

Step 4: Log in with a user account (or log in with a security token):

```
coreInfo = PictureparkService.Login(coreInfo);
```

Step 5: If the login was successful, you can now call any of the other Webservices, e.g. GetMetadata:

AssetMetadata assetMetadata = PictureparkService.GetMetadata(param1, param2, ..);
Please note: The code examples below are in C# and formatted as in VisualStudio and with Service Reference Settings configured as to reuse types in referenced assemblies.

The definition of complex data types depends on your proxy code generator, if you use one. Otherwise you must construct the complex data types on your one. In these latter cases, your code will differ from the examples. For example, if you reuse types in referenced assemblies in VisualStudio, the assignment of a DateTimeOffset data type looks like this:

```
CreationDate = System.DateTimeOffset.Now;
```

Without reusing of types, the same assignment looks like this:

```
CreationDate = new PictureparkService.DateTimeOffset()
{
    DateTime = System.DateTimeOffset.Now.DateTime,
    OffsetMinutes = (short)System.DateTimeOffset.Now.Offset.Minutes
};
```

3.4 Authentication (Webbrowser)

In order to be able to search and select any asset stored in Picturepark you need to login the user first. Here, you can provide SSO authentication with the Security Token method or have the user login manually through the Picturepark user interface provided in a later step.

3.5 Manual Authentication

If you do not create and later submit a Security Token or if that security Token is not valid the user is prompted to enter his Picturepark login credentials. He will then enter his e-mail address and password in the Picturepark form directly presented through the web browser.

3.6 Security Token Authentication

Get a security token from Picturepark for the user who is logged in at the external system by passing his e-mail address and optionally creating that user in Picturepark if not existing:

```
getSecurityToken(customerId, authenticationUser, authenticationUserPassword, emailAddress, userData, traceJobId)
```

`The authenticationUser` is an existing Picturepark user that must have the administrative privileges in Picturepark to create new users and assign them to user groups. This user is authenticated with his `authenticationUserPassword`. As a best practice advice the technical authentication user shouldn't be used later for downloading the files since he might have no access privileges on the assets. `emailAddress` is the e-mail address of the user logged into the external system, now being authenticated against Picturepark. If that `emailAddress` user does not already have a valid Picturepark user account it can be created by passing the `userData` defined by Picturepark (e.g. Address, City, Departement, LastName). The `traceJobId` is needed for tracing purposes only and can be null. See the Picturepark Documentation for further details about `getSecurityToken`, or the code example at the end of this documentation.

Example:

```
int[] userGroupIds = new int[] {3,17};

UserData userData = new UserData()
{

     EmailAddress = "max@mustermann.com",
     FirstName    = "Max",
     LanguageId   = (int)ApplicationLanguage.English,
     LastName     = "Mustermann",
     City         = "Aarau",
     Zip          = "5000",
     Company      = "VIT AG",
     CountryId    = 41,
     UserGroupIds = userGroupIds
};

string authenticationUser         = "adminuser@picturepark.com";
string authenticationUserPassword = "adminuser-password";
string emailAddress               = "max@mustermann.com";
int? traceJobId                   = null;

string securityToken = client.GetSecurityToken(customerId, clientGuid,
authenticationUser, authenticationUserPassword, emailAddress, userData, traceJobId);


CoreInfo ci = client.LoginWithSecurityToken(coreInfo, clientGuid, securityToken);
```

Note:

- If the user don`t exist he will be created in this example and he is assigned to the user groups with the Id`s 3 and 17.
- If the user is already existing in Picturepark he will be updated depending on the delivered user data object
- If the user should not be updated, e.g. if the user rights will be changed manually in Picturepark it is necessary to check if the user already exists in Picturepark. If so, the user data object can be null.

4 Asset ConnectorConnector

The external system has now all information needed for downloading the files from Picturepark. The Asset Connector can be called in a separate window. If the authentication is done by the external system the user has directly access to the assets. There are two options for downloading assets.

4.1 Open Asset Connector

Using the properties BaseUrl, CustomerID and SecurityToken you can call the AssetConnector component that has an URL like this:

```
https://customer.picturepark.com/Website/Publisher.aspx?Page=AssetConnector&SecToken=SecurityToken&redirect=RedirectUrl
oder https://customer.picturepark.com/Site/AssetConnector
```

`Customer` is replaced with the Picturepark short customer name used in the URL when accessing the system with your browser (e.g. VIT from http://vit.picturepark.com).

`SecurityToken` is replaced with the Picturepark SecurityToken received in the authentication step.

`RedirectUrl` is the return value address of the external system to which AssetConnector delivers a JSON object. This JSON object contains the properties as described below and must be decoded on behalf of the external system. The JSON response looks like:

```
[{"AssetId":1365,"DerivativeDefinitionId":6,"FileSizeInBytes":24849},{"AssetId":1365,"DerivativeDefinitionId":2,"FileSizeInBytes":22428}]
```

4.2 Filter Assets

In the example above the Asset Connector is called without any parameters to pre-select assets, which means that all assets of the system will be displayed. There are several possibilities to filter assets which should be displayed for the user.

4.2.1 Channels / Area

Picturepark channels are virtual partitions that can be used to provide multiple taxonomies and multitenant DAM services. Channels appear as tabs along the top of the Picturepark Core UI. Access to a channel is determined by user permissions. If the Picturepark system on which you call the Asset Connector has more than one Channel please choose the channel at the Asset Connector call.
Example:

```
https://customer.picturepark.com/Website/Publisher.aspx?Page=AssetConnector&area=1
```

4.2.2 Asset Type / File Type

An asset in Picturepark is an object which contains the original file of a specific file type, like jpeg, psd, video, pdf, audio file or word document. The file types are referenced to as asset types based on default definitions and necessary rendering actions and are defined in the database. The file type mapping is based on standard definitions. E.g. if you want only display images you need to query for the asset Type Id of images in the system and then you can attach this parameter to the Asset Connector call. This works via direct search which is explained in 4.2.4

You can find the asset type mapping at the Picturepark Management Console -> Derivatives -> Read Support.

Example:

In this example we are searching for all assets which have the AssetTypeId “2”.

```
https://customer.picturepark.com/Website/Publisher.aspx?Page=AssetConnector&ExS={"T":"Or","Val":[{"O":"StringContains","F":"AssetTypeId","V1":"2"}]}
```

4.2.3 Asset Container

An asset can be assigned to one or multiple categories a/o asset classes which can be browsed by users and are referenced to as asset container / asset class. Categories are shown as folder structure but should not to be confused with folders on the file system as they are more of a taxonomy structure where each folder acts as a keyword with a set of metadata. Asset classes are conditional tags which can trigger metadata for assigned assets (e.g. Asset class “stock image” enables metadata fields “expiry date” and “usage restriction” whereas asset class “portrait” only enables metadata fields “name”, “title”, “department”)
You can add the parameter AssetContainerId to the AssetConnector call to display only assets from a specific asset container.

Example:

```
https://customer.picturepark.com/Website/Publisher.aspx?Page=AssetConnector&AssetContainerId=132
```

4.2.4 Direct search of assets

You can add search parameters to the Asset Connector call to filter very detailed which assets should be displayed for the user.

Below is an example of a URL structure which contains search parameters. Variables are marked green, values blue.

```
http://customer.picturepark.com/Website/Publisher.aspx?Page=AssetConnector&ExS={"T":"Or","Val":[{"O":"StringListEqual","F":"AssetName","V1":"test"},{“O”:"NumericLargerThan",F:"AssetId",V1:123}]}
```

This search is searching for assets which contain “test” in the AssetName OR have an AssetId larger than 123. Please note the following rules for entering values:

- Quotation marks for text and lists are mandatory. E.g. for a single value for the AssetId there is no need for quotation marks (see example above)
- Several values can be listed by using a comma

Examples:

The following examples are in addition to the following URL:

```
http://customer.picturepark.com/Website/Publisher.aspx?Page=AssetConnector&
```

1. Finds assets which have the asset name „Test.jpg“

```
ExS={"T":"Or","Val":[{"O":"StringListEqual","F":"AssetName","V1":"Test.jpg"}]}
```

2. Finds assets which contain „JPG“ in the asset name

```
ExS={"T":"Or","Val":[{"O":"StringListContains","F":"AssetName","V1":"JPG"}]}
```

3. Finds assets which have an asset id larger than 123 in the channel with the id 2

```
ExS={"T":"Or","Val":[{"O":"NumericLargerThan","F":"AssetId","V1":"123"}]}&Area=2
```

4. Finds assets which have the asset id 123, 234 or 345

```
ExS={"T":"Or","Val":[{"O":"NumericListEqual","F":"AssetId","V1":"123,234,345"}]
```

5. Finds all assets which have the asset id 123 or 234 or “Logo” as asset name

```
ExS={"T":"Or","Val":[{"O":"NumericListEqual","F":"AssetId","V1":"123,234"},{"O":"StringListEqual","F":"AssetName","V1":"Logo"}]}
```

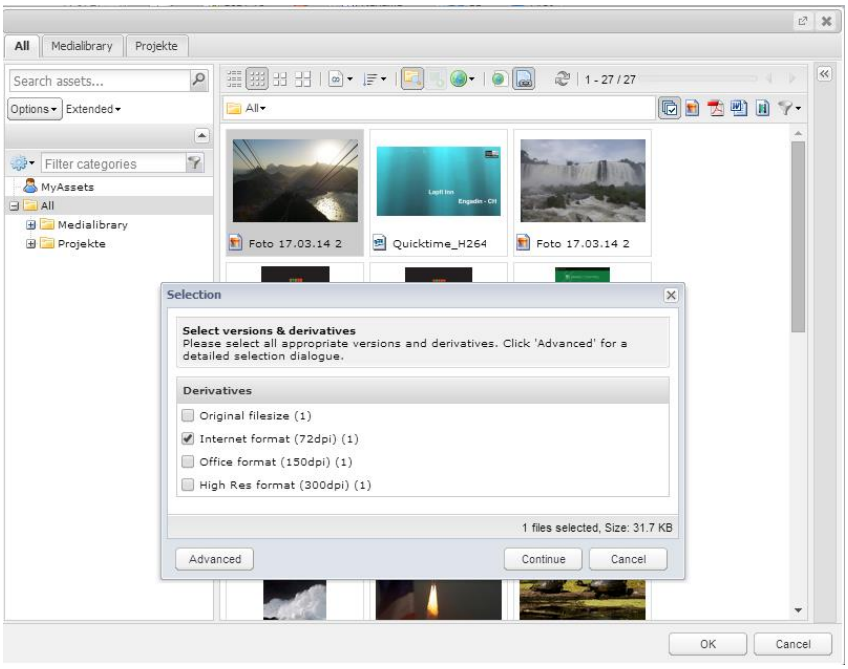
6. Finds all assets which have an asset id between 123 and 234

```
ExS={"T":"Or","Val":[{"O":"NumericBetween","F":"AssetId","V1":123,"V2":234}]}
```

4.3 Possibilities for the user

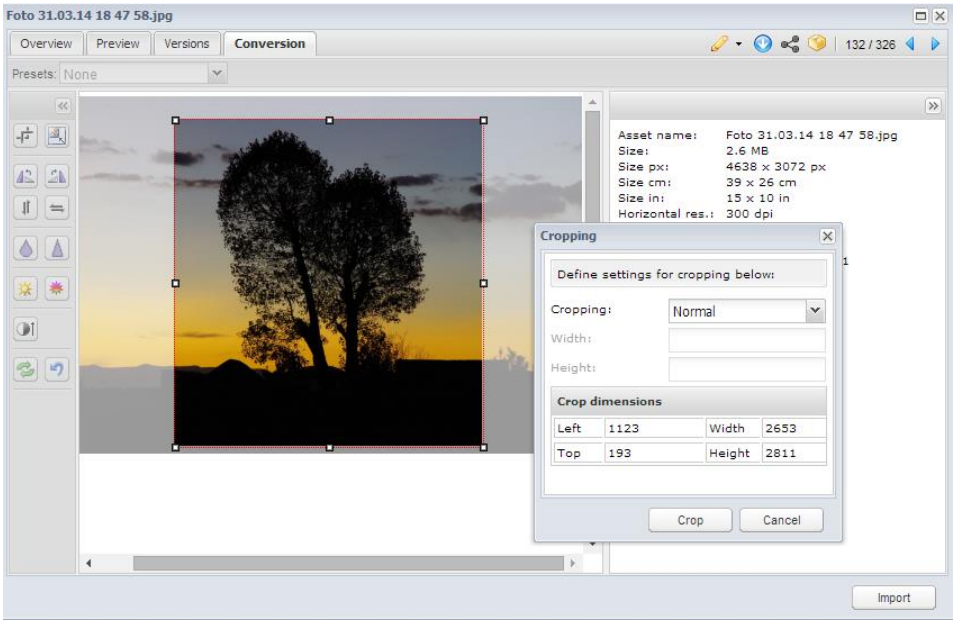
4.3.1 Choose the Original asset or a desired derivative.

Choose one or multiple assets and click “Ok” for download.



4.3.2 Convert the asset before downloading

Choose one asset, double click, and then got to conversion, work on the image and click “Import”.



4.4 Download

4.4.1 Download one or multiple assets (original or derivatives)

After the Picturepark AssetConnector has send the array which contains the assets to the redirect URL the external system can download the assets.

For the download the following information are needed:

- Asset Id
- Derivative Id

The download method:

```
Download download = PictureparkService.Download(coreInfo, assetSelection, downloadOptions);
```

Parameters:

`coreInfo` please see 3.3 (Create Session)

```
List<AssetSelection> assetSelection = new List<AssetSelection>();
assetSelection.Add(new AssetSelection() {AssetId = 1, DerivativeDefinitionId = 2});
assetSelection.Add(new AssetSelection() {AssetId = 4, DerivativeDefinitionId = 6});
DownloadOptions downloadOptions = new DownloadOptions()
{
  CreateZip4SingleFile = false,
  UsagePurpose         = "Just a test by the programmer",
  UserAction           = UserAction.DerivativeDownload
};
```

4.4.2 Download converted asset

If the user has made a conversion of the asset, the response from the Asset Connector looks like this:

http://link-to-thrid-partysystem.domain.com/?assets=[{%22AssetId%22:905,%22DerivativeDefinitionId%22:null,%22DerivativeId%22:null,%22FilePath%22:%22Download.aspx?Purpose=ImageRenderingManager&RelativePath=\\eee7b4a2-2a37-42bc-8cd4-2c99b12d37b3\\Foto%2012.04.14%2009%2031%2056.jpg%22,%22FileSizeInBytes%22:1125618}]

With this response it is possible to create the direct download link and download the converted asset. Example:

http://link-to-thrid-partysystem.domain.com/Website/Download.aspx?Purpose=ImageRenderingManager&RelativePath=\\eee7b4a2-2a37-42bc-8cd4-2c99b12d37b3\\Foto%2012.04.14%2009%2031%2056.jpg

Note: The converted asset is stored in the temp folder of the system. In Picturepark it is possible to setup a “Clean up” job. This will normally run every night and deletes all files in the temp folder.

5 Other Webservice possibilities around the Picturepark Asset Connector

5.1 Get Meta data

After the asset is downloaded to the external system it is possible to download also the Meta data of the asset. For this the following steps must be done:

1. Get all Meta data fields from the system to map them via the Id to the values of the Meta data array of the asset.

   ```
   PictureparkService.GetAssetFields(coreInfo);
   ```

2. Get the asset by the asset Id

   ```
   PictureparkService.GetAssetsByIds(coreInfo, assetIds, assetFilter);
   ```

3. Get the Meta data array for this asset

   ```
   PictureparkService.GetAssetsMetadata(coreInfo, assetIds, fieldIds);
   ```

5.2 Create asset Links instead of download the assets

If the assets should be linked instead of imported to the third party system this option is the right choice. Used for example to stream a video directly from Picturepark. This works similar to the steps described in 4.2.1, but at the end the external system will not download the asset.

This is helpful for large data objects like videos. Picturepark can deliver the video within a video player. The code can directly placed as an iFrame. The video will be streamed from Picturepark. Example:

```
<iframe width="560" height="315" src="https://poc382.picturepark.com/embed/XUndE2sD/508/2"
frameborder="0" allowfullscreen></iframe>
```

A direct link to an asset within an asset link consists of some parts:

- URL
- Link token as a clearly identifier
- Distinction if it is a Download (D) or a View (V) Link. Note: A view Link displays an asset directly in the browser. If it is a video it is embedded in a JavaScript video player.
- The asset Id. A Link can contain different Assets. Via the Id it is possible to display certain asset in the browser.
- Derivate Definition ID

Asset Link Example:
https://sei.picturepark.com/Go/gA1LzC9E/

Direct View Link Example:
https://sei.picturepark.com/Go/gA1LzC9E/V/910/1

Content of asset links can be changed in Picturepark, e.g. if an asset within the link is remove or created. Then the Asset Id changes. That means that the external System need to update the Asset Id within the link to the Asset in the external System. If a user updates an asset the link will not change.

Create an Asset Link:

```
Create an asset link for a collection of asset derivatives.

List<AssetSelection> assetSelection = new List<AssetSelection>();
assetSelection.Add(new AssetSelection() { AssetId = 512, DerivativeDefinitionId = 7 });
assetSelection.Add(new AssetSelection() { AssetId = 507, DerivativeDefinitionId = 8 });
assetSelection.Add(new AssetSelection() { AssetId = 505, DerivativeDefinitionId = 6 });

DateTimeOffset expirationDate = new DateTimeOffset(2020, 7, 31, 0, 0, 0, TimeSpan.Zero);
string assetLinkName          = "Images Collection 2014";
string assetLinkDescription   = "Contains pictures about the latest collection.";

BusinessProcessShort assetLink = PictureparkService.CreateAssetLink(coreInfo, customerId,
assetLinkName, assetLinkDescription, assetSelection, expirationDate);
```

5.3 **Further possibilities**

Picturepark public Webservice, especially the method `getAssets()`, provides you with a lot of information of the asset which you can then transfer to the object stored in the external system, e.g. taking over the title, copyright information etc. from the Picturepark asset. For instance, an external system such as a WCMS could transfer the title of the image as well and place this as the alt tag.

Additionally, you can implement data lifecycle validations e.g. by storing in your external system the hash/MD5 checksum, file size or last changed datetime-stamp. This information can then be used for a daily job which calls Picturepark and loops through all Picturepark-supplied asset in the external system, checking if an update to this file has been or the file is no longer available and a consequent manual re-sync should be executed, triggered by a notification message sent to the external system admin.

6 Picturepark documentation

VIT will provide you with an account for their partner portal from where you can access detailed technical Webservice documentation.

Additionally, VIT or certified partners will also provide you with a short reference covering all customer-specific system configuration parameters you need to know for your integration.