Dynamic Media with OpenAPI capability - Assets Selector Integration

nv1.0.2

Assumptions

- Customer
  - Customer is already entitled to AE/AEM Assets CS (Cloud Service) and the **Dynamic Media with OpenAPI** capability
    - Customers' developers are aware of the **Dynamic Media with OpenAPI** docs. See https://adobe-aem-assets-delivery-experimental.redoc.ly/

Functional Flow

Step #	Persona	Activity
1.	Dam Author/Librarian	Uploads assets in the AE/AEM Assets CS
2.	Brand Manager	Reviews the assets, and according marks them 'Approved'/'Rejected'.
3.	Developer (who consumes Brand approved assets using	Builds the solution which integrates the **Dynamic Media with OpenAPI** capabilities with Asset Selector for the Experience Author to build their web properties.

	the **Dynamic Media with OpenAPI** API host)	
4.	Experience Author	Experience author uses the solution/integration which the developer built, to author their web properties, like site pages, mails etc.

Above illustrates the flow in a sequence, but **Dynamic Media with OpenAPI** capabilities, in conjunction with AEM Assets CS environments, allows for asset governance, and asset upload, review and solution development can happen constantly and in parallel.

Version History

Date	Version	Change (from earlier version)
15 April 2023	1.0	1st version
20 May 2023	1.0.1	Trivial refactoring and verbiage updates
30 Jul 2023	1.0.2	Fixes mention of incorrect API for integration
01 Feb 2024	1.0.3	• Steps to configure selection of single or multiple assets • Updated assets-selectors.js source URL • Defining AEM Tier Type with Delivery and enabling repository selection

Asset Selector Integration with Adobe IMS Sign-up/Sign-In

The following steps are to set up Asset Selector, where users would be prompted to sign up or log on to the Adobe IMS (Identity Management Service) with their respective accounts in a modal popup.

Folder/Resource Setup

```
Asset_Selector
  |
  | -- index.html
  |
  | -- cert.pem
  |
  | -- csr.pem
  |
  | -- key.pem
```

cert.pem, csr.pem, key.pem are needed for SSL (Secure Sockets Layer) based interaction. These files won't be needed if the hosting service already has SSL enabled, for e.g., on AEMaaCS.
You could use mkcert utility to generate the self-signed certificates and keys to enable SSL, if the hosting service does not already have SSL enabled. Note that you may have to trust the self-signed certificates to avoid SSL warnings in browsers

Code setup guidelines (within index.html)

```
<!-- Include the CDN (Content Delivery Network) link in your script tag -->
<script src="https://experience.adobe.com/solutions/CQ-assets-selectors/static-assets/resources/assets-selectors.js"></script>
<script>
    function handleSelection(selectedAssets) {
        // selectedAssets is an array of assets being selected
    }
    let imsInstance = null;
    let imsProps = {
        imsClientId: "<IMS Client Id>", // please refer Appendix Item (6) for IMS Client creation
        imsScope: "openid",
        redirectUrl: window.location.href, // the host of the URL (Universal Resource Locator) should be specified in the "redirect URL patterns" of the IMS client.
        // modalMode governs whether the IMS login screen opens in a modal popup or within the container HTML (Hyper Text Markup Language) loading tab.
        modalMode: true
    }

    function load() {
        const registeredTokenService = PureJSSelectors.registerAssetsSelectorsAuthService(imsProps);
        imsInstance = registeredTokenService;
    };
    // Must be loaded before the selectors are rendered.
    load();
    const assetSelectorProps = {
        aemTierType: ['delivery'], // Please note this is an array.
```

```
        imsOrg: 'xxxx@AdobeOrg', // TODO: replace with IMS Org Id
        hideTreeNav: true,
        handleSelection, // call back invoked on selection
        ...otherProps // other props to be passed to the AssetSelector component
    };

    // Use the PureJSSelectors in global to render the AssetSelector component
    const container = document.getElementById('asset-selector-container'); //
container should be an HTML Element and not a jQuery Element.
    PureJSSelectors.renderAssetSelectorWithAuthFlow (container,
assetSelectorProps);
</script>
```

Running the setup

The container HTML – index.html, needs to be hosted and be served over HTTPS. More specifically, opening the index.html simply in browser would not work.

Information received in selection call-back

Upon selection an array of selected objects are received, each element corresponding to each selected asset from the Asset Selector.

PS: the "…" is a placeholder for other pieces of information, but the following (in the format below) is sufficient to generate approved assets' delivery URL

Sharing below, sample schema of 1 object, from the array of objects expected to be received upon selection.

```
{
  "dc:format": "image/jpeg",
  "repo:assetId": "urn:aaid:aem:888ac66a-5dc4-4459-9b4d-bd26a9020b44",
  "repo:name": "accenture-7.jpg",
  "repo:repositoryId": "delivery-p47604-e144858.adobeaemcloud.com",
  ...
}
```

Above information can be used to **construct approved assets' delivery URLs**, as per below

Approved assets' delivery API specification: https://adobe-aem-assets-delivery-experimental.redoc.ly

Approved assets' delivery capability – Asset Delivery

URL format:

```
https://<delivery-api-host>/adobe/dynamicmedia/deliver/<asset-id>/<seo-
name>.<format>?<image-modification-query-parameters>
```

e.g.

```
https://delivery-p12345-
e654321.adobeaemcloud.com/adobe/dynamicmedia/deliver/urn:aaid:aem:d12a69a1-
406a-41e8-b133-86743b83c971/accenture-
2.jpg?width=319&height=319&preferwebp=true
```

Where:

- Host: https://delivery-p12345-e654321.adobeaemcloud.com
- API root: "/adobe/dynamicmedia/deliver"
- <asset-id>: Asset identifier

- <seo-name>: name of the asset
- <format>: output format
- <image modification query parameters>: as support by the approved assets' delivery API specification

How to construct the Dynamic Delivery URL from the selection (selected asset)

- Selected asset, as held in the `handleSelection` callback, is a JSON object, say `assetJsonObj`.
- The URL can be formed by concatenating the below, and aligning with the format as explained above

Host	`assetJsonObj["repo:repositoryId"]`
API Root	`/adobe/dynamicmedia/deliver`
asset-id	`assetJsonObj["repo:assetId"]`
seo-name	`assetJsonObj["repo:name"].split(".").slice(0,-1).join(".")`
format	`.jpg`

Approved assets' delivery API – Original Asset binary Delivery

URL format:

```
https://<delivery-api-host>/adobe/assets/deliver/<asset-id>/<seo-name>
```

e.g.,

```
https://delivery-p12345-
e654321.adobeaemcloud.com/adobe/assets/deliver/urn:aaid:aem:41b4d2b7-bb8a-4d91-
bc30-a848a9ebcad8/football.pdf
```

Where:

- Host: https://delivery-p12345-e654321.adobeaemcloud.com
- API root for Original Rendition Delivery: "/adobe/assets/deliver"
- <asset-id>: asset identifier
- <seo-name>: name of the asset (may or may not have an extension)

How to construct the Original Asset binary delivery URL from the selection (selected asset)

- Selected asset, as held in the `handleSelection` callback, is a JSON object, say `assetJsonObj`.
- The URL can be formed by concatenating the below, and aligning with the format as explained above

Host	`assetJsonObj["repo:repositoryId"]`
API Root	`/adobe/assets/deliver`
asset-id	`assetJsonObj["repo:assetId"]`
seo-name	`assetJsonObj["repo:name"]`

Appendix

Asset Selector Setup with valid IMS token integration guidelines

```
// Sample Code

<script>
const apiToken="<valid IMS token>";

function handleSelection(selection) {
console.log("Selected asset: ", selection);
};

function renderAssetSelectorInline() {
console.log("initializing Asset Selector");
const props = {
"repositoryId": "delivery-p64502-e544757.adobeaemcloud.com",
"apiKey": "ngdm_test_client",
"imsOrg": "<IMS org>",
"imsToken": apiToken,
handleSelection,
hideTreeNav: true
}
const container = document.getElementById('asset-selector-container');
PureJSSelectors.renderAssetSelector(container, props);
}
$(document).ready(function() {
renderAssetSelectorInline();
});
</script>
```

Steps to customize Filter Panel

Add the below to the `assetSelectorProps` object from the same shared.

```
filterSchema: [
    {
        header: 'File Type',
        groupKey: 'TopGroup',
        fields: [
            {
                element: 'checkbox',
                name: 'type',
                options: [
                    {
                        label: 'Images',
                        value: '<comma separated mimetypes, without space, that
denote all images, for e.g., image/>',
                    },
                    {
                        label: 'Videos',
                        value: '<comma separated mimetypes, without space, that
denote all videos for e.g., video/,model/vnd.mts,application/mxf>'
                    }
                ]
            }
        ]
    },
    {
        fields: [
            {
```

```
                element: 'checkbox',
                name: 'type',
                options: [
                    { label: 'JPG', value: 'image/jpeg' },
                    { label: 'PNG', value: 'image/png' },
                    { label: 'TIFF', value: 'image/tiff' },
                    { label: 'GIF', value: 'image/gif' },
                    { label: 'MP4', value: 'video/mp4' }
                ],
                columns: 3,
            },
        ],
        header: 'Mime Types',
        groupKey: 'MimeTypeGroup',
    }},
    {
        fields: [
            {
                element: 'checkbox',
                name: 'property=metadata.application.xcm:keywords.value',
                options: [
                    { label: 'Fruits', value: 'fruits' },
                    { label: 'Vegetables', value: 'vegetables'}
                ],
                columns: 3,
            },
        ],
        header: 'Food Category',
        groupKey: 'FoodCategoryGroup',
    }

 ],
```

Steps to customize information viewed in modal popup, on selecting info icon of assets in card view

```
// Create an object infoPopoverMap and set the property `infoPopoverMap` with it
in assetSelectorProps


const infoPopoverMap = (map) => {
    // for e.g., to remove `path` from the info popover view
    let defaultPopoverData = PureJSSelectors.getDefaultInfoPopoverData(map);

    return defaultPopoverData.filter((i) => i.label !== 'Path'

  };

assetSelectorProps.infoPopoverMap = infoPopoverMap;
```

Steps to register callbacks related to IMS Service interaction

```
// object `imsProps` to be defined as below
let imsProps = {
    imsClientId: <IMS Client Id>, // please refer Appendix Item (6) for IMS
Client creation

    imsScope: "openid",
```

```
    redirectUrl: window.location.href,
    modalMode: true,
    adobeImsOptions: {
        modalSettings: {
            allowOrigin: window.location.origin,
        },
        useLocalStorage: true,
    },
    onImsServiceInitialized: (service) => {
        console.log("onImsServiceInitialized", service);
    },
    onAccessTokenReceived: (token) => {
        console.log("onAccessTokenReceived", token);
    },
    onAccessTokenExpired: () => {
        console.log("onAccessTokenError");
        // re-trigger sign-in flow
    },
    onErrorReceived: (type, msg) => {
        console.log("onErrorReceived", type, msg);
    },

}
```

Enable drag and drop mode

```
// add the following properties to `assetSelectorProps`

rail: true,
acvConfig: {
    dragOptions: {
        allowList: {
            '*': true,
        },
    },
    selectionType: 'multiple'
}
```

```
// the drop handler to be implemented

function drop(e) {
    e.preventDefault();

    // following helps you get the selected assets - an array of objects.
    const data = JSON.parse(e.dataTransfer.getData('collectionviewdata'));
}
```

Steps to configure selection of single or multiple assets

```
// add the following properties to `assetSelectorProps`

acvConfig: {
    selectionType: 'multiple' // ‘single’ for single selection
}

// the `handleSelection` callback, always gets you the array of selected assets
```

Asset Selector setup with fixed repository

```
<!-- Include the CDN (Content Delivery Network) link in your script tag -->
<script src="https://experience.adobe.com/solutions/CQ-assets-selectors/static-assets/resources/assets-selectors.js"></script>
<script>
    function handleSelection(selectedAssets) {
        // selectedAssets is an array of assets being selected
    }
    let imsInstance = null;
    let imsProps = {
        imsClientId: "<IMS Client Id>", // please refer Appendix Item (6) for IMS Client creation
        imsScope: "openid",
        redirectUrl: window.location.href, // the host of the URL (Universal Resource Locator) should be specified in the "redirect URL patterns" of the IMS client.
        // modalMode governs whether the IMS login screen opens in a modal popup or within the container HTML (Hyper Text Markup Language) loading tab.
        modalMode: true
    }

    function load() {
        const registeredTokenService = PureJSSelectors.registerAssetsSelectorsAuthService(imsProps);
        imsInstance = registeredTokenService;
    };
    // Must be loaded before the selectors are rendered.
    load();
    const assetSelectorProps = {
        repositoryId: 'delivery-pxx-eyy.adobeaemcloud.com', // TODO: replace with AEM Assets CS environment's host name
        imsOrg: 'xxxx@AdobeOrg', // TODO: replace with IMS Org Id
        hideTreeNav: true,
        handleSelection, // call back invoked on selection
        ...otherProps // other props to be passed to the AssetSelector component
    };

    // Use the PureJSSelectors in global to render the AssetSelector component
    const container = document.getElementById('asset-selector-container'); // container should be an HTML Element and not a jQuery Element.
    PureJSSelectors.renderAssetSelectorWithAuthFlow (container, assetSelectorProps);
</script>
```

IMS Client ID / API Key Provisioning

Please contact the engineering contacts (mentioned at the end of this document) to get an IMS Client ID (which will be used as API Key).

Adobe contacts

1. apogupta@adobe.com (Product Leader)
2. arora@adobe.com (Engineering Leader)
3. chiki@adobe.com (Lead Engineer)