**INTERGRATE MX PLATFORM IN MEMBER PORTAL**

For integration there are two options that MX provides:

1. Connect widget
2. Use of several API’s and build your own UI.

**Workflow with the Connect widget:**

1. Creating a user with a request to the MDX Real Time API.
2. Loading the Connect widget in verification mode on your website or within a WebView.
3. End users create a member, launch a verification job, and respond to MFA within the Connect widget.
4. Getting an API token with the SSO API.
5. Getting a session token with the Nexus API.
6. Calling the identify endpoint if needed.
7. Check the member's verification status with the Nexus API answering MFA if necessary.
8. Once verification has completed, getting the verification data with another request to the Nexus API. (Fetch Account Number)

**Workflow with APIs only**

1. Creating a user (MDX Real Time API).
2. Getting an API token for that user (SSO API).
3. Getting a session token for that user (Nexus API).
4. Getting institution information (Nexus API).
5. Get institution-required credentials (Nexus API).
6. Creating a member with the correct institution code and credentials required by that institution (Nexus API).
7. Calling the verify endpoint (Nexus API).
8. Polling the member's verification status (Nexus API).
9. Answering MFA if necessary (Nexus API).
10. Reading the member's AccountNumbers (Nexus API).

For our project we will opt for using The Connect widget since it simplifies creating members and answering multi-factor authentication while providing a ready-made user interface for these verification steps.

**Prerequisites:**

Before we start integrating with the user below are some of key steps that needs to be implemented to support MX integration

**Creation of Accessors to fetch API key**: - Accessors represent the partner who accesses one or more of MX products. Accessors have an associated API key which must be used in all API requests to MX as a layer of authentication. IP whitelisting is also managed through accessors.

Accessors can be created only by MX.

**Creation of Clients to fetch client id**: - Clients also represent partners that interact with the MX platform, but one level down in the platform's hierarchy. An accessor can have one or multiple clients.

Clients have an associated client\_id which is used in API requests as an added layer of authentication. Clients are also where MX products are customized and branded to meet the needs of a specific partner. Clients can be created only by MX.

**IP whitelisting**: - We must provide a list of IP addresses that have to be whitelisted. MX will reject requests that do not come from whitelisted IP addresses. Conversely, we must whitelist MX's IP addresses in cases where MX makes calls to their system, such as with MDX On Demand or webhooks.

**TLS**: - All requests to the MX platform must be made using TLS 1.2 or higher with known-secure ciphers. The MDX Real Time API is served over HTTPS. To ensure data privacy, unencrypted HTTP requests are not supported.

**Base URL**

All URL endpoints in the MDX Real Time API have a base. The domain of the base URL will depend on the environment being addressed:

Integration Server (for initial integration testing)

https://int-live.moneydesktop.com/:client\_id

Production Server

https://live.moneydesktop.com/:client\_id

The client ID of the client the user belongs to is also specified in the base URL of all requests, as in the following example:

<https://live.moneydesktop.com/Payflex/>

**CONNECT WIDGET IMPLEMENTATION**

1. **Create a user:**

This is the first step of creation of user in MX platform.

Below are the fields required for creation of user. Any fields described in the user fields section may optionally be included in this request.

|  |  |
| --- | --- |
| Id | Unique partner-provided identifier for the user. |
| Guid | The MX-generated guid for the user. |
| Birthdate | Birthdate of the user. |
| credit\_score | Credit score of the user. |
| Email | Email address of the user. |
| first\_name | First name of the user. |
| gender | TRUE |
| is\_disabled | Determines if a user is disabled from logging in. true or false. Defaults to false. |
| last\_name | Last name of the user. |
| logged\_in\_at | Date and time the user last logged in. |
| metadata | Additional information a partner can store on the user. |
| Phone | Phone number of the user. |
| zip\_code | Zip code of the user. |

**Sample Code: -**

public class MDX\_RealTime

{

private String BaseUrl, ClientId, ApiKey;

public MDX\_RealTime(String BaseUrl, String ClientId, String ApiKey)

{

this.BaseUrl = BaseUrl;

this.ClientId = ClientId;

this.ApiKey = ApiKey;

}

public static void main(String[] args)

{

Root root = InitializeMember();

String base\_url = "https://int-live.moneydesktop.com";

String client\_id = ":client\_id";

String api\_key = ":api\_key";

MDX\_RealTime mdx = new MDX\_RealTime(base\_url, client\_id, api\_key);

String user\_id = "U-39XBF7";

Task task = mdx.CreateUserAsync(root);

}

private static Root InitializeMember()

{

return new Root()

{

user = new User()

{

birthdate = "1959-07-17",

credit\_score = "718",

email = "walterwhite@example.com",

first\_name = "Walter",

gender = "MALE",

id = "U - 39XBF7",

last\_name = "White",

metadata = "Additional Information",

phone = "5055551234",

zip\_code = "87101",

},

};

}

public async Task CreateUserAsync(Root user)

{

try

{

string url = BaseUrl + "/" + ClientId + "/users.json";

using (var httpClient = new HttpClient())

{

using (var request = new HttpRequestMessage(new HttpMethod("POST"), url))

{

request.Headers.TryAddWithoutValidation("Accept", "application/vnd.moneydesktop.mdx.v5+json");

request.Headers.TryAddWithoutValidation("MD-API-KEY", ApiKey);

var jsonobject = JsonConvert.SerializeObject(user);

request.Content = new StringContent(jsonobject);

request.Content.Headers.ContentType = MediaTypeHeaderValue.Parse("application/vnd.moneydesktop.mdx.v5+json");

var response = await httpClient.SendAsync(request);

}

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

}

public class User

{

public string birthdate { get; set; }

public string credit\_score { get; set; }

public string email { get; set; }

public string first\_name { get; set; }

public string gender { get; set; }

public string id { get; set; }

public string last\_name { get; set; }

public string metadata { get; set; }

public string phone { get; set; }

public string zip\_code { get; set; }

}

public class Root

{

public User user { get; set; }

}

1. **Load the Connect widget:**

Next will be to integrate the connect widget.

There are two methods for loading the Connect widget:

1. getting a URL through the SSO API
2. using the widget loader script.

In either case, the Connect widget must be loaded with the mode configuration option set to verification.

Use the wait\_for\_full\_aggregation option as well. This delays the member connected postMessage until after the verification process is totally complete. This makes it easier for partners to keep the widget open for the correct period of time with the minimum number of API calls.

**Option A — Use the SSO API**

The endpoint described below also supports JSON encoding for both requests and responses.

(This is not a recommended usage)

**Sample Code: -**

using (var httpClient = new HttpClient())

{

using (var request = new HttpRequestMessage(new HttpMethod("POST"), "https://int-sso.moneydesktop.com/:client\_id/users/:id/urls.xml"))

{

request.Headers.TryAddWithoutValidation("Accept", "application/vnd.moneydesktop.sso.v3+xml");

request.Headers.TryAddWithoutValidation("MD-API-KEY", ":api\_key");

request.Content = new StringContent("<url>\n <type>connect\_widget</type>\n <mode>verification</mode>\n <wait\_for\_full\_aggregation>true</wait\_for\_full\_aggregation>\n </url>");

request.Content.Headers.ContentType = MediaTypeHeaderValue.Parse("application/vnd.moneydesktop.sso.v3+xml");

var response = await httpClient.SendAsync(request);

}

}

**Option B — Use the widget loader**

* **Get a URL**

First, get a URL as described in option A above.

* **Add the widget loader script to the page**

The widgets are loaded onto the page through a custom javascript file we refer to as the "widget loader". This file can be placed anywhere on the page, but as we'll discuss in the next steps, it should be placed before any other code related to the widgets.

* **Add a widget placeholder element**

Next, we need to add a placeholder element that the widget loader will use to embed an iframe containing the widget. The element must have an id of md-widget.

* **Load the widget**

Next, we must tell the widget loader to load the widget into the placeholder element. We do this by creating a new instance of the MoneyDesktopWidgetLoader class, which is defined in the widget loader script.

When you instantiate MoneyDesktopWidgetLoader, you must pass in an object with at least the required url parameter and possibly one or more optional parameters.

The widget loader will wait until the page has been loaded and then load the widget into the placeholder element. The resulting HTML on the page will look similar to the example on the right.

**Sample Code in HTML: -**

<html>

<head>

<title>My Web Page</title>

<script type="text/javascript" src="https://widgets.moneydesktop.com/assets/mx-widgetloader.js"></script>

<script type="text/javascript">

var myWidget = new MoneyDesktopWidgetLoader();

</script>

</head>

<body>

<div id="md-widget' class='md-widget-loaded">

<iframe width="850" height="550" border="0" frame="0" frameborder="0" allowtransparency="true" src="https://widgets.moneydesktop.com/md/connect/XXXXX" marginheight="0" marginwidth="0"></iframe>

</div>

</body>

</html>

1. **Load the Connect widget:**

The connect widget handles the creation of a member as well as the verification and MFA processes.

We need to listen for a member updated postMessage event from the Connect widget which will provide both the necessary member\_guid and the current connection\_status. If your implementation uses WebViews, you'll need to listen for MX postMessage alternative for WebViews.

When the connection\_status value is CONNECTED, you can move on to step 4.

Or, alternatively, if the end user closes the Connect widget early or some other problem occurs before you see a CONNECTED status, you can simply use the member\_guid to immediately check the member's verification status via Nexus as in step 4.

You may also choose to set up a member webhook which will deliver the information you need as the member's state changes.

1. **Get an API token:**

This endpoint returns an api\_token which can then be used to open a Nexus session. The user\_id must match the id specified when creating the user.

Each api\_token is one-time use and expires in ten minutes. A fresh api\_token must be requested each time a Nexus session is initiated.

**Sample Code: -**

using (var httpClient = new HttpClient())

{

using (var request = new HttpRequestMessage(new HttpMethod("GET"), "https://int-sso.moneydesktop.com/:client\_id/users/:user\_id/api\_token.xml"))

{

request.Headers.TryAddWithoutValidation("Accept", "application/vnd.moneydesktop.sso.v3+xml");

request.Headers.TryAddWithoutValidation("MD-API-KEY", ":api\_key");

var response = await httpClient.SendAsync(request);

}

}

1. **Open a Nexus session**

Use this endpoint to get a session token. This session token must be sent as a header with each Nexus request. Each Nexus session is opened in the context of a single user associated with that session token.

When using the verification endpoints the create session request body should contain "skip\_aggregation":true, as shown in the example below. This will prevent automatic aggregation being initiated for the user's existing members when the session is opened. We can then choose to aggregate the existing member, or to proceed directly to the verify member endpoint.

**Sample Code: -**

using (var httpClient = new HttpClient())

{

using (var request = new HttpRequestMessage(new HttpMethod("POST"), "https://int-data.moneydesktop.com/sessions"))

{

request.Headers.TryAddWithoutValidation("MD-API-TOKEN", "VVCFDQY3Iig3ODM3LTlmNDItMmY1OS03NDZhLWIwNTcwNDc2NGRiNXwzZjg1MTQ4YzAyMThkZjI2OTI2ZjFjYmJhNmY1MzU2MTI0MThiNTIyOGVjYjg4YTRjNjljODJhYTQ3MWMwMTQyZGY5ZmM1OGQ2YmU1M2ZiMjNlOWZhZTE3Y2MxNjU4YWYxYmE0NjRiMDg3Nzk4N2U0YzU5ZTE1NjM1MDUwYjEzMTkzYzYyMzhiYjY3MDhmNjEyOGIzNjIyMDIwMmMzZTIzfFlVam1oSmhaOHNMMGVlMTg0YWhBODdmZWlVRmh4YlNKVGhVZUM4RVl5QXc=");

request.Headers.TryAddWithoutValidation("Accept", "application/vnd.mx.nexus.v1+json");

request.Content = new StringContent("{\"session\":{\"skip\_aggregation\":true}}");

request.Content.Headers.ContentType = MediaTypeHeaderValue.Parse("application/vnd.mx.nexus.v1+json");

var response = await httpClient.SendAsync(request);

}

}

1. **Check the member's verification status**

Several fields on the member give important information about the state of the verification including connection\_status, is\_being\_aggregated, successfully\_aggregated\_at. These fields are also used for standard aggregation jobs, hence the reference in their names.

A connection status of CONNECTED means that the member was successfully authenticated, and verification has begun. The is\_being\_aggregated field will tell you whether the verification job has completed; the field will be true while verification is taking place and returns to false when verification is complete. The successfully\_aggregated\_at field will tell you the exact time that the verification job has completed.

You may need to repeatedly poll this endpoint until either an end state or an actionable state is reached.

Depending on the exact state of the member, partners may need to re-load the Connect widget with a specific set of options in order to resolve an actionable member state.

**Sample Code: -**

using (var httpClient = new HttpClient())

{

using (var request = new HttpRequestMessage(new HttpMethod("GET"), "https://int-data.moneydesktop.com/members/MBR-7aa13bdb-2866-ba38-b326-d9fb32268f9b"))

{

request.Headers.TryAddWithoutValidation("MD-SESSION-TOKEN", "CWforZl1Vn2vC\_v6H4rnQRT1DoWpDouJAV-\_5TBmiQRAtA8rsOG\_BoajTiOSsL0A3bd-bmHXlA-eQzc9ywItKg");

request.Headers.TryAddWithoutValidation("Accept", "application/vnd.mx.nexus.v1+json");

var response = await httpClient.SendAsync(request);

}

}

1. **Read the account numbers**

If MX has *both an account number and a routing number* for at least one of the member's accounts, that information will be returned when calling *read account numbers*. No information will be returned for accounts that are missing a value for one or both of these fields.

**Sample Code: -**

using (var httpClient = new HttpClient())

{

using (var request = new HttpRequestMessage(new HttpMethod("GET"), "https://int-data.moneydesktop.com/members/MBR-c6536f4c-61dc-a047-5c45-ee38b8ba812a/account\_numbers"))

{

request.Headers.TryAddWithoutValidation("MD-SESSION-TOKEN", ":session\_token");

request.Headers.TryAddWithoutValidation("Accept", "application/vnd.mx.nexus.v1+json");

var response = await httpClient.SendAsync(request);

}

}