



# intalio

## Intalio IAM Developer Guide (EN)

Version: 3.8.0

Intalio is an international software provider with 30 years of innovation in the field of Content and Process Services. By combining cutting-edge technologies with AI and Machine Learning to introduce advanced Line of Business solutions, Intalio managed to become a trusted leader in Digital Transformation across the US, Europe, and the MENA region.

### **Vision.**

Making business simple by connecting people, processes and content.

# Table of Contents

<b>1. INTEGRATION</b>	<b>4</b>
1.1. Data Sources	4
Notes	5
Properties	5
1.2. Jobs	6
1.2.1. User Jobs	6
1.2.1.1. User Push job	6
1.2.1.2. User Pull Job	7
1.2.1.3. Application Access	8
Notes	8
1.2.2. Structure Jobs	8
1.2.2.1. Structure Push Job	8
1.2.2.2. Structure Pull Job	10
1.2.3. Attribute Mapping	10
1.3. Code	11
1.3.1. User Push Job	11
1.3.1.1. Find Method	11
1.3.1.2. Create Method	12
Note	12
1.3.1.3. Update Method	13
1.3.2. User Pull Job	14
1.3.2.1. Count Method	14
1.3.2.2. List Method	14
Notes	15
1.3.3. Structure Push Job	17
1.3.3.1. Find Method	17
1.3.3.2. Create Method	18
Note	18
1.3.3.3. Update Method	19
1.3.4. Structure Pull Job	20
1.3.4.1. Count Method	20
1.3.4.2. List Method	20
Notes	21
<b>2. VALIDATOR</b>	<b>23</b>
Notes	23
Properties	24
2.2. Code	24

<b>3. WEB SERVICE LOGIN PROVIDER .....</b>	<b>26</b>
3.1. Code .....	27
3.1.1. <i>Fetch Method</i> .....	27
3.1.2. <i>Authenticate Method</i> .....	27
3.1.3. <i>Example</i> .....	28
<b>4. CUSTOM GRANT TYPE .....</b>	<b>29</b>
Notes     30	
4.1. Code .....	30
4.2. Using the custom grant type .....	31
<b>5. TWO-FACTOR AUTHENTICATION PROVIDER .....</b>	<b>32</b>
5.1. Code .....	33
5.1.1. <i>SendOTP</i> .....	33
Note     34	
5.1.2. <i>ValidateOTP</i> .....	34
5.1.3. <i>Example</i> .....	35
<b>6. CUSTOM SOLUTION .....</b>	<b>36</b>
Create a Solution .....	36
Add a Reference to Intalio.IAM.Core.dll .....	38
Add New Class(es) .....	38
Final Step .....	39
Notes     40	
Hints     40	

# 1. Integration

This section covers the details of the data sources and jobs. Job sequence and administration under the integration menu are not within the scope of this document, their details are covered in the admin guide.

## 1.1. Data Sources

The data sources are available under the integration menu.

To add a new data source, click on the new button, the following form will open:

Name \*

User

Class for push ⓘ

Choose File

Class for pull ⓘ

Choose File

Assembly fully qualified name for push ⓘ

Assembly fully qualified name for pull ⓘ

Structure

Class for push ⓘ

Choose File

Class for pull ⓘ

Choose File

Assembly fully qualified name for push ⓘ

Assembly fully qualified name for pull ⓘ

Properties

Name	Default value	Description	Mandatory	Encrypted
			<input type="checkbox"/>	<input type="checkbox"/>

Fill data source name and the following (at least one should be filled):

### User section:

- If this data source will be used in a user **push** job, fill either class for push or assembly fully qualified name for push:
  - **Class for push:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationUserPush`
  - **Assembly fully qualified name for push:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationUserPush`

- If this data source will be used in a user **pull** job, fill either class for pull or assembly fully qualified name for pull:
  - **Class for pull:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationUserPull`
  - **Assembly fully qualified name for push:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationUserPull`

#### Structure section:

- If this data source will be used in a structure **push** job, fill either class for push or assembly fully qualified name for push
  - **Class for push:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationStructurePush`
  - **Assembly fully qualified name for push:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationStructurePush`
- If this data source will be used in a structure **pull** job, fill either class for pull or assembly fully qualified name for pull
  - **Class for pull:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationStructurePull`
  - **Assembly fully qualified name for pull:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.IIntegrationStructurePull`

## Notes

- At least one of the classes or assembly fully qualified names should be filled
- If the class uses any external library, the library DLL should be uploaded in IAM's **Assembly** section (check the admin guide)
- Assembly fully qualified name format:  
`{namespace}.{classname}, {dll name}, Version={assembly version}, Culture=neutral, PublicKeyToken={null or the public key token if exist}`
- Error that might occur:  
`System.IO.FileLoadException: The given assembly name or codebase was invalid. (0x80131047)`  
If this error occurred, please rewrite your assembly fully qualified name. This might occur if the value is copied and pasted and sometime on copy, characters like comma or space gets modified and will not be recognized when pasted.

## Properties

If your jobs (your custom classes) need some properties or configuration whose values are dynamic or that may differ from a job to another, you can specify these properties in the **Properties** section (details on how they are used in the code are explained in the [code](#) section):

- **Name:** Property name.
- **Default value:** defines the default value of the property if any.
- **Description:** defines the description of the property.
- **Mandatory:** check this checkbox if the property is mandatory and should be filled in the job.
- **Encrypted:** check this checkbox if the property value should be encrypted in the jobs and the database.

These properties will be available for each of the data source's jobs.

## 1.2. Jobs

### 1.2.1. User Jobs

The user jobs are available under integration -> user jobs menu.

#### 1.2.1.1. User Push job

Add new user job, the following form will open:

The screenshot shows a web form for configuring a 'User Push' job. At the top, there are three icons: a database icon, a document icon, and a calendar icon. The form is divided into several sections:

- Name \***: A text input field containing 'Push Database table users'.
- Data source \***: A dropdown menu with 'Database' selected.
- Primary key ⓘ**: A dropdown menu with 'Select primary key' selected.
- Application**: A dropdown menu with 'Select application' selected.
- Operation**: Two radio buttons, 'Push' (selected) and 'Pull'.
- Permissions**: Four checkboxes, all of which are checked: 'All users', 'Allow create', and 'Allow update'.
- Properties**: A section containing:
  - DataBase Type \* ⓘ**: A text input field containing 'sql'.
  - Connection String \***: A text input field containing 'Server=ETGS-mal\SQL14;Database=testIAM;MultipleActiveResultSi'.
  - Table \***: A text input field containing 'Test1'.
- \* Required fields**: A note at the bottom left.
- Next**: A green button at the bottom right.

Fill/Edit the job's configuration:

- **Name:** job name.
- **Data Source:** select the related data source.

- **Primary key:** if any, select from the IAM attributes dropdown the attribute that will hold the external application primary key value.  
Supported primary key attribute types: String, Short, Integer, long.
- Check **Push**.
- Either select **All Users** or select an application from the **Application** dropdown:
  - **All Users:** will push all IAM users.
  - **Application:** will push only the IAM users having access to the selected application.
- **Allow create:** check if the job should create new users.
- **Allow update:** check if the job should update existing users.
- **Properties:** the data source properties will be available here to fill their values.

Click next, fill the [attributes mapping](#) and set a schedule if needed.

### 1.2.1.2. User Pull Job

Add new user job, the following form will open:

The form is titled 'User Pull Job' and contains the following fields and options:

- Name \***: Text input field containing 'Pull Database table users'.
- Data source \***: Dropdown menu showing 'Database'.
- Primary key**: Dropdown menu showing 'Select primary key'.
- Push/Pull**: Radio buttons for 'Push' (unchecked) and 'Pull' (checked).
- Allow create**: Checkmark (checked).
- Allow update**: Checkmark (checked).
- Properties**:
  - DataBase Type \***: Text input field containing 'sql'.
  - Connection String \***: Text input field containing 'Server=ETGS-mal\SQL14;Database=testIAM;MultipleActiveResultSi'.
  - Table \***: Text input field containing 'Test1'.
- \* Required fields**: Legend for required fields.
- Next**: Button at the bottom right.

Fill/Edit the job's configuration:

- **Name:** job name.
- **Data Source:** select the related data source.
- **Primary key:** if any, select from the IAM attributes dropdown the attribute that will hold the

external application primary key value.

Supported primary key attribute types: String, Short, Integer, long.

- Check **Pull**.
- **Allow create**: check if the job should create new users.
- **Allow update**: check if the job should update existing users.
- Properties: the data source properties will be available here to fill their values.

Click next, set the [application access](#).

Click next, fill the [attributes mapping](#) and set a schedule if needed.

### 1.2.1.3. Application Access

The screenshot shows the 'Edit' form for Application Access. It features a progress bar with four steps: Application (selected), Role, User type, and a final step. Below the progress bar, there are three dropdown menus: 'Application' (with 'Select application' text), 'Role' (with 'Select role' text), and 'User type' (with 'Select user type' text). To the right of these is a green '+' button. Below the dropdowns are two yellow warning boxes: 'Users exceeding application license limit will be deactivated' and 'This step will be ignored when updating an existing user'. At the bottom right are 'Previous' and 'Next' buttons.

Here, you can set the permissions to be given to the pulled users on specific application(s) by specifying the application, role and user type.

### Notes

- If the number of pulled users (new users not already existing in IAM that will be created) exceeds the selected application user license limit, these users will be pulled to IAM but will be deactivated.
- If a user already exists in IAM, on update, this step will be ignored to not override any action the admin might have taken on the user application access.

## 1.2.2. Structure Jobs

The structure jobs are available under integration -> structure jobs menu.

### 1.2.2.1. Structure Push Job

Add new structure job, the following form will open:



New

Name \*

Data source \*

Primary key ?

Select primary key

Structure type

All

Structures

☒ Push ☐ Pull

☐ All structures

☐ Allow create

☐ Allow update

Properties

Connection String \*

DataBase Type \* ?

sql

\* Required fields

Next

Fill/Edit the job's configuration:

- **Name:** job name.
- **Data Source:** select the related data source.
- **Primary key:** if any, select from the IAM attributes dropdown the attribute that will hold the external application primary key value.  
Supported primary key attribute types: String, Short, Integer, long.
- **Structure type:**
  - All: will push all structures.
  - Internal: will push only internal structures.
  - External: will push only external structures.
- Check **Push**.
- Either select **All Structures** or select root structures from the **Structures** dropdown:
  - **All Structures:** will push all IAM structures.
  - **Structures:** will push only the selected root structures and their children.
- **Allow create:** check if the job should create new structures.
- **Allow update:** check if the job should update existing structures.
- **Properties:** the data source properties will be available here to fill their values.

Click next, fill the [attributes mapping](#) and set a schedule if needed.

### 1.2.2.2. Structure Pull Job

Add new structure job, the following form will open:

Fill/Edit the job's configuration:

- **Name:** job name.
- **Data Source:** select the related data source.
- **Primary key:** if any, select from the IAM attributes dropdown the attribute that will hold the external application primary key value.  
Supported primary key attribute types: String, Short, Integer, long.
- **Structure type:**
  - Internal: the pulled structures will be created as internal structures.
  - External: the pulled structures will be created as external structures.
- Check **Pull**.
- **Allow create:** check if the job should create new structures.
- **Allow update:** check if the job should update existing structures.
- **Properties:** the data source properties will be available here to fill their values.

Click next, fill the [attributes mapping](#) and set a schedule if needed.

### 1.2.3. Attribute Mapping

The attribute mapping is used to map between IAM attributes and your application attributes.

**Name:** IAM attribute name

**Mapping name:** name of the attribute in your application

The selected attributes will be available to your custom class with their mapping name.

<input type="checkbox"/>	Name	Mapping name
<input checked="" type="checkbox"/>	First name	<input type="text" value="Fname"/>
<input type="checkbox"/>	Middle name	<input type="text"/>
<input checked="" type="checkbox"/>	Last name	<input type="text" value="Lname"/>
<input type="checkbox"/>	Email	<input type="text"/>
<input type="checkbox"/>	Gender	<input type="text"/>

## 1.3. Code

In this section the integration job classes are explained. For details on how to create a solution and a class in .Net you can check the [custom solution](#) section.

**Note:** if your custom job did not work properly or an error occurred, you can check Hangfire (Integration > Administration) or IAM logs (Settings > logs) for errors or warnings.

### 1.3.1. User Push Job

Add new class implementing the interface **IIntegrationUserPush**.

When implementing the interface, 3 methods will be created: Create, Find, Update.

The push job will call the [Find](#) method first, then the [Create](#) or [Update](#) method.

#### 1.3.1.1. Find Method

```
public bool Find(IntegrationItem integrationItem)
```

A parameter **IntegrationItem** will be passed to the **Find** method. The IntegrationItem consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue:** the primary key value of the user we are trying to find and push.
  - IAM's user id if no primary key is selected.
  - Attribute value if primary key is selected.

In this method you can implement your needed criteria to find the user in your application.

This method should return:

- **true** if the user was found in the application (already pushed)
- **false** if the user was not found in the application (not pushed yet)

### 1.3.1.2. Create Method

```
public IntegrationItem Create(IntegrationUserItem integrationUser)
```

A parameter **IntegrationUserItem** will be passed to the **Create** method. The **IntegrationUserItem** consists of the following properties:

- **ParameterCollection**: Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName**: the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue**: the primary key value of the user we are trying to find and push.
  - IAM's user id if no primary key is selected.
  - Attribute value if primary key is selected.
- **UserModel** containing the user's info:
  - **Id**: IAM's user id.
  - **FirstName**: first name.
  - **MiddleName**: middle name.
  - **LastName**: last name.
  - **FullName**: full name.
  - **Email**: email.
  - **ApplicationRole**: the user's role name in the selected application. (If an application is selected in the job and not all users)
  - **DefaultStructureId**: user's default structure id.
  - **ManagerId**: manager id.
  - **Groups**: the list of groups the user is in.
  - **StructureIds**: the list of structure ids the user is in.
  - **Structures**: the list of structures the user is in.
  - **Roles**: the user's RoleModel in the selected application. (If an application is selected in the job and not all users)
  - **Attributes**: the list of the job's mapped attributes with their values related to the user.

In this method you can implement your needed criteria to create the user in your application.

This method should return an **IntegrationItem**.

#### Note

If in the job, an attribute is selected as the primary key, the **PrimaryKeyValue** in the returned item must be filled. (This value will be saved in the user's attribute and sent the next time the push job is executed)

The **PrimaryKeyValue** can be returned empty only if no primary key is selected in the job's

dropdown.

### 1.3.1.3. Update Method

```
public IntegrationResult Update(IntegrationUserItem integrationUser)
```

A parameter **IntegrationUserItem** will be passed to the **Update** method. The **IntegrationUserItem** consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if none is selected.
- **PrimaryKeyValue:** the primary key value of the user we are trying to find and push.
  - IAM's user id if no primary key is selected.
  - Attribute value if primary key is selected.
- **UserModel** containing the user's info:
  - **Id:** IAM's user id.
  - **FirstName:** first name.
  - **MiddleName:** middle name.
  - **LastName:** last name.
  - **FullName:** full name.
  - **Email:** email.
  - **ApplicationRole:** the user's role name in the selected application. (If an application is selected in the job and not all users)
  - **DefaultStructureId:** user's default structure id.
  - **ManagerId:** manager id.
  - **Groups:** the list of groups the user is in.
  - **StructureIds:** the list of structure ids the user is in.
  - **Structures:** the list of structures the user is in.
  - **Roles:** the user's RoleModel in the selected application. (If an application is selected in the job and not all users)
  - **Attributes:** the list of the job's mapped attributes with their values related to the user.

In this method you can implement your needed criteria to update the user in your application.

This method should return an **IntegrationResult**:

- result: bool value indicating if the user was updated or not
- message: string

## 1.3.2. User Pull Job

Add new class implementing the interface **IIntegrationUserPull**.

When implementing the interface, 2 methods will be created: Count, List.

The pull job will call the [Count](#) method first, then the [List](#) method.

### 1.3.2.1. Count Method

```
public int Count(IntegrationItem integrationItem)
```

A parameter **IntegrationItem** will be passed to the **Count** method. The IntegrationItem consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue:** will be empty.

In this method you can implement your needed criteria to get the count of the users that will be pulled from your application.

This method should return an integer indicating the total number of users to be pulled.

### 1.3.2.2. List Method

```
public List<IntegrationUserModel> List(int startIndex, int pageSize, IntegrationItem integrationItem)
```

Parameters passed to the **List** method:

- **startIndex:** starting record - The number of records to skip before starting to return records from the query.
- **pageSize:** number of records to be returned (15).
- **IntegrationItem:** consists of the following properties:
  - **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
  - **PrimaryKeyName:** the primary key attribute name.
    - name of the attribute selected from the job's primary key dropdown.
    - "Id" if no primary key is selected.
  - **PrimaryKeyValue:** will be empty.

In this method you can implement your needed criteria to get the list of the users that will be pulled from your application.

This method should return **List<IntegrationUserModel>**. **IntegrationUserModel** containing the user's info:

- **Id:** IAM's user id.
- **UserName:** username, only fill it in the model or attributes mapping if login provider is local, AD or Web Service, otherwise leave it empty.
- **FirstName:** first name.
- **LastName:** last name.
- **Email:** email.
- **UserApplicationRole:** used to set the user's applications access. List<ApplicationRole> containing RoleId, ApplicationId, UserTypeId.

To get the active application ids check the database or the APIs **ListActiveApplications** and **ListActiveApplicationsIds** under Intalio.IAM.Core.API.ManageApplication.

To get the role ids check the database or the APIs **ListAllRoles** and **ListAllRolesIds** under Intalio.IAM.Core.API.ManageRole.

You can check the file **Intalio.IAM.chm** included in the IAM package for all the APIs provided by IAM.

- **LoginProviderId:** the login provider id.  
Check the database or IAM APIs available in the file **Intalio.IAM.chm**
- **GroupIds:** used to set the user's groups. List<short> containing the IAM groups ids.  
To get the group ids check the database or IAM APIs available in the file **Intalio.IAM.chm**
- **Attributes:** List<StringValueText> the list of user's attributes having as **Text** the attribute mapping name (name of the attribute in your application) and **Value** the value of this attribute.

## Notes

- On pull, IAM will check if the user already exists. If exists, it will be updated, otherwise it will be created. How IAM will check if the user exists:
  - If primary key is selected in the job dropdown, IAM will try to find the user by this attribute value. This attribute must be returned in the attributes list.  
Example: Add an attribute "ExternalId" in IAM. Select this attribute in the job as primary key:
    - If this attribute is NOT selected in the mapping, it should be returned in the IntegrationUserModel **Attributes** with text "ExternalId"
    - If this attribute is selected in the mapping with mapping name for example "ExternalIdName", it should be returned in IntegrationUserModel **Attributes** with text "ExternalIdName"
  - If primary key is NOT selected, IAM will try to find the user by his IAM id, therefore the **Id** in the IntegrationUserModel must be filled in this case.
  - If no primary key is selected in the job, and no id is sent in the IntegrationUserModel, IAM will try to find the user by his username or email depending on the login provider type. In this case **LoginProviderId** must be specified, if not it will be considered as local user and search for it by username.

- **First name, last name** and **email** are mandatory for new users. If they are not filled in the user model nor in the mapping and are provided null or empty the user will be skipped and not created.
- On pull, IAM checks if the user already exists. If not found, the user is created based on the login provider id sent.
  - For Local, Active directory and web service users, the username must be provided and if another user already has the same username, the user will be skipped and not created.
  - For OIDC and social login providers, the email must be provided and unique, if another user already has the same email, the user will be skipped and not created.
  - For social login providers, if the email does not match the provider domain(s), the user will be skipped and not created.
- If Login provider id is not provided, new users will be added as local users. Default login provider id is 1 = local
- For local users (new users created), a random password will be generated. The password can be changed by the user by clicking on the forget password link.
- Skipped users, if any, will be shown in IAM logs (Level = Job)
- For UserApplicationRole, if a certain application role was not added to the user, it could be one the following reasons:
  - Application does not exist (wrong application id or application is disabled)
  - Role does not exist (wrong role id)
  - UserType does not exist (wrong id different than 1 for “Editor” or 2 for “ReadOnly”)
  - Multiple UserApplicationRole are sent with the same application id (only 1 role will be added for a specific application)
- If the UserType specified in the UserApplicationRole exceeds the application’s user license limit, the related user will be created in IAM but will be deactivated.
- If no UserApplicationRole is specified for a user, the job will check its [application access](#).
- On update, UserApplicationRoles are ignored and won’t be updated to not override any changes that might be done by the admin in IAM.
- On update, LoginProviderId is ignored as IAM does not support changing login provider for users.
- On update, if the email is changed, the user will be skipped and not updated if the new email already exists for another user or if the user’s login provider is a social login provider and the new email does not match the provider domain(s).
- If gender is mapped, the value of this attribute should be filled with one character either “M” or “F”.
- If StructureId and ManagerId are mapped and returned in the attributes of the pull job make sure they contain their IAM id not your application’s id otherwise it will cause issues in IAM. (If you need to update them)
- **Active** attribute represents the user status. On pull, if selected in the mapping:



- On create: the user will be active or not based on this attribute value unless the user was assigned a role to an application that exceeds its license limit, if so, it will be deactivated on creation.
  - On update: active user can be deactivated. But if the user was deactivated it cannot be activated from the job, this can be done only by the admin from the IAM interface.
- For GroupIds:
  - If the list is empty:
    - New users will be added with no groups
    - Existing users: the user's groups will not be updated. He will remain in his existing groups, if they are no longer required, the admin must remove this user from his groups via the IAM interface.
  - If the list contains groups:
    - New users will be added to the groups
    - Existing users: the user's groups will be modified. He will be removed from any existing groups and added to the sent ones.  
If he should remain in one or more of his existing groups, these groups should be included in the GroupIds list.
- You can check the file **Intalio.IAM.chm** included in the IAM package for all the APIs provided by IAM.

### 1.3.3. Structure Push Job

Add new class implementing the interface **IIntegrationStructurePush**.

When implementing the interface, 3 methods will be created: Create, Find, Update.

The push job will call the [Find](#) method first, then the [Create](#) or [Update](#) method.

#### 1.3.3.1. Find Method

```
public bool Find(IntegrationItem integrationItem)
```

A parameter **IntegrationItem** will be passed to the **Find** method. The IntegrationItem consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue:** the primary key value of the structure we are trying to find and push.
  - IAM's structure id if no primary key is selected.
  - Attribute value if primary key is selected.

In this method you can implement your needed criteria to find the structure in your application.

This method should return:

- **true** if the structure was found in the application (already pushed)
- **false** if the structure was not found in the application (not pushed yet)

### 1.3.3.2. Create Method

```
public IntegrationItem Create(IntegrationStructureItem integrationStructure)
```

A parameter **IntegrationStructureItem** will be passed to the **Create** method. The **IntegrationStructureItem** consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue:** primary key value of the structure we are trying to find and push.
  - IAM's structure id if no primary key is selected.
  - Attribute value if primary key is selected.
- **ParentPrimaryKeyValue:** if an attribute is selected from the job's primary key dropdown, this property will contain the attribute value of the parent structure.

If no primary is selected, it will be empty as the parent structure IAM's id is provided in the structure model.

- **StructureModel** containing the structure's info:
  - **Id:** IAM's structure id.
  - **Name:** structure name.
  - **Code:** structure code.
  - **ManagerId:** structure manager id (IAM id).
  - **ParentId:** IAM's structure parent id.
  - **External:** bool – true if external, false if internal.
  - **Attributes:** the list of the job's mapped attributes with their values related to the structure.
  - **Parent: ParentStructureModel** (id, name, code, external, attributes of the parent structure) (not provided – will be null)

In this method you can implement your needed criteria to create the structure in your application.

This method should return an **IntegrationItem**.

### Note

If in the job, an attribute is selected as the primary key, the **PrimaryKeyValue** in the returned item must be filled. (This value will be saved in the structure's attribute and sent the next time the push job is executed)

The **PrimaryKeyValue** can be returned empty only if no primary key is selected in the job's dropdown.

### 1.3.3.3. Update Method

```
public IntegrationResult Update(IntegrationStructureItem integrationStructure)
```

A parameter **IntegrationStructureItem** will be passed to the **Update** method. The **IntegrationStructureItem** consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue:** the primary key value of the structure we are trying to find and push.
  - IAM's structure id if no primary key is selected.
  - Attribute value if primary key is selected.
- **ParentPrimaryKeyValue:** if an attribute is selected from the job's primary key dropdown, this property will contain the attribute value of the parent structure.

If no primary is selected, it will be empty as the parent structure IAM's id is provided in the structure model.

- **StructureModel** containing the structure's info:
  - **Id:** IAM's structure id.
  - **Name:** structure name.
  - **Code:** structure code.
  - **ManagerId:** structure manager id (IAM id).
  - **ParentId:** IAM's structure parent id.
  - **External:** bool – true if external, false if internal.
  - **Attributes:** the list of the job's mapped attributes with their values related to the structure.
  - **Parent: ParentStructureModel** (id, name, code, external, attributes of the parent structure) (not provided – will be null)

In this method you can implement your needed criteria to update the structure in your application.

This method should return an **IntegrationResult**:

- result: bool value indicating if the structure was updated or not.
- message: string.

## 1.3.4. Structure Pull Job

Add new class implementing the interface **IIntegrationStructurePull**.

When implementing the interface, 2 methods will be created: Count, List.

The pull job will call the [Count](#) method first, then the [List](#) method.

### 1.3.4.1. Count Method

```
public int Count(IntegrationItem integrationItem)
```

A parameter **IntegrationItem** will be passed to the **Count** method. The IntegrationItem consists of the following properties:

- **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
- **PrimaryKeyName:** the primary key attribute name.
  - name of the attribute selected from the job's primary key dropdown.
  - "Id" if no primary key is selected.
- **PrimaryKeyValue:** will be empty.

In this method you can implement your needed criteria to get the count of the structures that will be pulled from your application.

This method should return an integer indicating the total number of structures to be pulled.

### 1.3.4.2. List Method

```
public List<StructureModel> List(int startIndex, int pageSize, IntegrationItem integrationItem)
```

Parameters passed to the **List** method:

- **startIndex:** starting record - The number of records to skip before starting to return records from the query.
- **pageSize:** number of records to be returned (15).
- **IntegrationItem:** consists of the following properties:
  - **ParameterCollection:** Dictionary<string,string> containing the job properties and their value.
  - **PrimaryKeyName:** the primary key attribute name.
    - name of the attribute selected from the job's primary key dropdown.
    - "Id" if no primary key is selected.
  - **PrimaryKeyValue:** will be empty.

In this method you can implement your needed criteria to get the list of the structures that will be pulled from your application.

This method should return **List<StructureModel>**. **StructureModel** containing the structure's info:

- **Id**: IAM's structure id.
- **Name**: structure name.
- **Code**: structure code.
- **ManagerId**: leave empty not used in the pull job.
- **ParentId**: IAM's structure parent id.
- **External**: bool – ignored, the pulled structure type will be set as selected in the job configuration.
- **Attributes**: List<StringValueText> the list of structure's attributes having as **Text** the attribute mapping name (name of the attribute in your application) and **Value** the value of this attribute.
- **Parent**: null.

## Notes

- On pull, IAM will check if the structure already exists. If exists, it will be updated, otherwise it will be created. How IAM will check if the structure exists:
  - If primary key is selected in the job dropdown, IAM will try to find the structure by this attribute value. This attribute must be returned in the attributes list.  
Example: Add an attribute "ExternalId" in IAM. Select this attribute in the job as primary key:
    - If this attribute is not selected in the mapping, it should be returned in the StructureModel **Attributes** with text "ExternalId"
    - If this attribute is selected in the mapping with mapping name for example "ExternalIdName", it should be returned in StructureModel **Attributes** with text "ExternalIdName"
  - If primary key is NOT selected, IAM will try to find the structure by its IAM id, therefore the **Id** in the StructureModel must be filled in this case.
- If structure name or code already exists, structure will be skipped and not be created.
- If structure was found in IAM and its type does not match the select type in the job, the structure will be skipped and not updated.
- Skipped structures, if any, will be shown in IAM logs (Level = Job)
- For parent structure id:
  - If primary key is selected in the job dropdown, IAM will try to find the parent by its "StructureParentId" attribute value.  
**StructureParentId** attribute must be returned in the attributes list having as a value the structure id in your application.  
Example:
    - If this attribute is not selected in the mapping, it should be returned in the StructureModel **Attributes** with text "StructureParentId"

- If this attribute is selected in the mapping with mapping name for example “StructureParentExternalId”, it should be returned in StructureModel **Attributes** with text “StructureParentExternalId”
  - If primary key is NOT selected, IAM will try to find the parent by its IAM id, therefore the **ParentId** in the StructureModel must be filled in this case.
- You can check the file **Intalio.IAM.chm** included in the IAM package for all the APIs provided by IAM.

## 2. Validator

This section covers the details of the validator class. For more details about the validator, you can check the admin guide.

To add a new validator, click on the **new validator** button, the following form will open:

**New validator**

Name \*

Attribute types \*

Assembly fully qualified name

Class

Javascript

**Validator properties**

Name	Default value	Description	Mandatory	Is Number
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	<input type="checkbox"/>

\* Required fields

Fill the following:

- **Name:** validator name.
- **Attribute types:** select the attributes types that can use this validator.
- Either fill the assembly fully qualified name or upload a class.
  - **Class:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.IValidator`
  - **Assembly fully qualified name:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.IValidator`
- **JavaScript:** If the validator has a front-end validation, you can upload a JavaScript (.js) file containing your function. If a file is upload, you also need to specify the JavaScript function name that will be used for validation.

### Notes

- If the class uses any external library, the library DLL should be uploaded in IAM's **Assembly** section (check the admin guide)
- Assembly fully qualified name format:

{namespace}.{classname}, {dll name}, Version={assembly version}, Culture=neutral, PublicKeyToken={null or the public key token if exist}

- Error that might occur:

System.IO.FileLoadException: The given assembly name or codebase was invalid. (0x80131047)

If this error occurred, please rewrite your assembly fully qualified name. This might occur if the value is copied and pasted and sometime on copy, characters like comma or space gets modified and will not be recognized when pasted.

## Properties

If your validator (your custom class) needs some properties or configuration whose values are dynamic or may differ from an attribute using this validator to another, you can specify these properties in the **Properties** section (details on how they are used in the code are explained in the [code](#) section):

- **Name:** Property name.
- **Default value:** defines the default value of the property if any.
- **Description:** defines the description of the property.
- **Mandatory:** check this checkbox if the property is mandatory and should be filled in the attribute.
- **Is Number:** check this checkbox if the property value is a number.

These properties will be available for each of the attributes using this validator.

**Example:** A File extension validator can have property **FileExtensions**, which can have as value “.jpg,.png” for an attribute and “.docx,.xlsx” for another.

## 2.2. Code

In this section the validator class is explained. For details on how to create a solution and a class in .Net you can check the [custom solution](#) section.

A validator sample class **RangeValidator** is provided with this document.

**Note:** if your custom validator did not work properly or an error occurred, you can check the IAM logs (Settings > logs) for errors.

Add new class implementing the interface **IValidator**.

When implementing the interface, 1 method will be created: *IsValid*.

```
public ValidationResult IsValid(object value, Dictionary<string, string> properties);
```



Parameters passed to the method:

- **Value:** object containing the value entered by the user for the attribute using this validator.
- **Properties:** Dictionary<string,string> containing the validator properties and their values.

In this method you can implement your needed criteria to validate the user input.

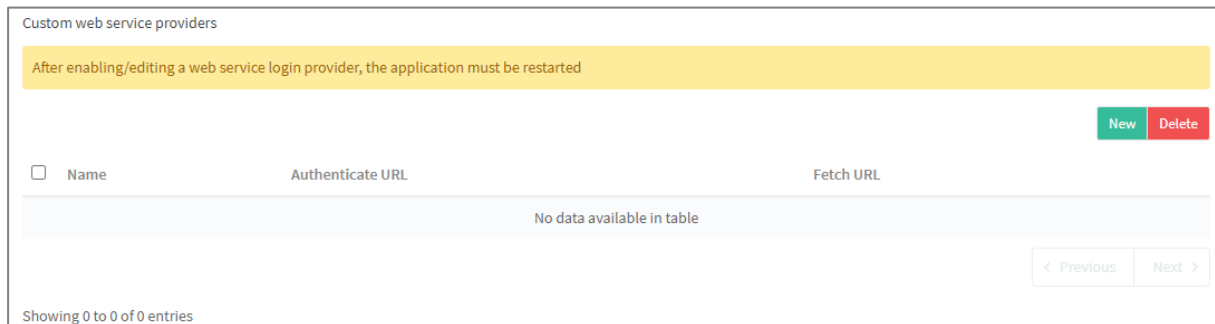
This method should return a **ValidationResult**:

- **Valid:** bool value indicating if the user's input value is valid or not.
- **ValidationMessage:** string.

### 3. Web Service Login Provider

This section covers the details of web service login provider. Other login providers are not within the scope of this document, their details are covered in the admin guide.

The Custom web service providers are available under the Login Providers menu.



Custom web service providers

After enabling/editing a web service login provider, the application must be restarted

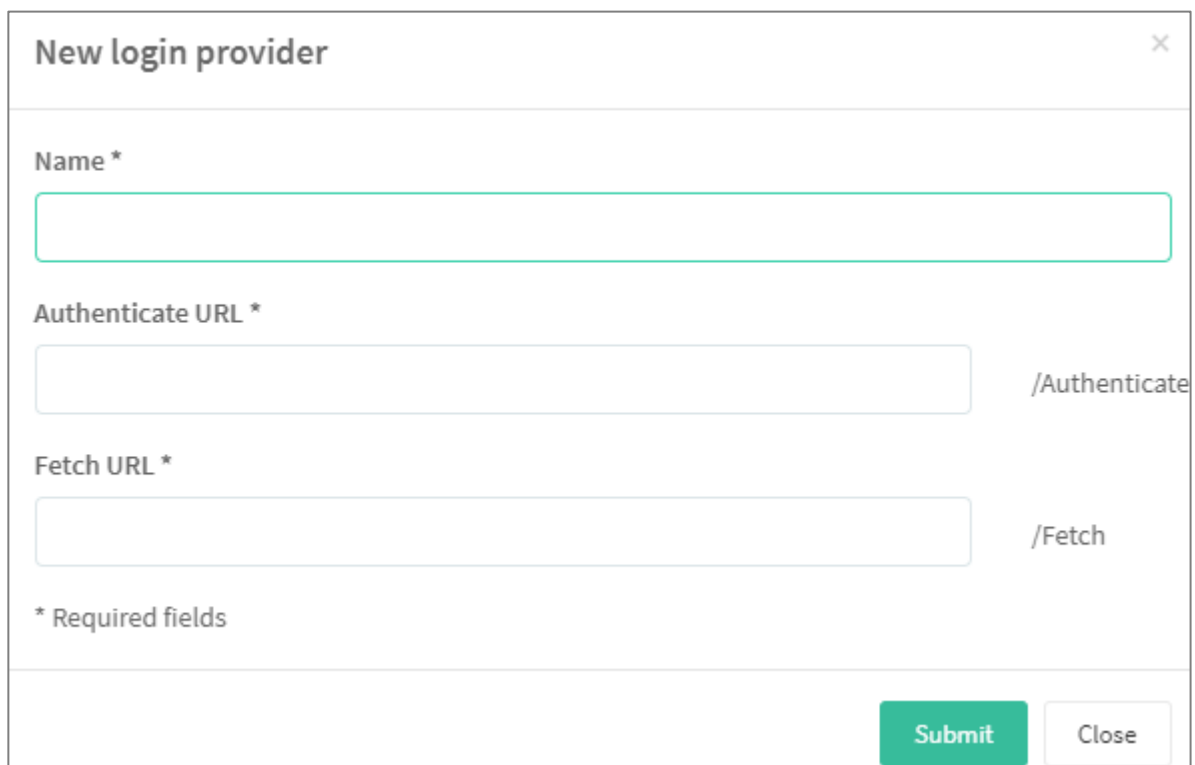
New Delete

<input type="checkbox"/>	Name	Authenticate URL	Fetch URL
No data available in table			

< Previous Next >

Showing 0 to 0 of 0 entries

To add a new provider, click on the **new** button, the following form will open:



New login provider

Name \*

Authenticate URL \*

Fetch URL \*

/Authenticate

/Fetch

\* Required fields

Submit Close

Fill the below:

- **Name:** web service login provider name.
- **Authenticate URL:** the URL of your custom web service that contains your **Authenticate** method.  
(http(s)://{servername:port/controller\_if\_any}/Authenticate)
- **Fetch URL:** the URL (http(s)://{servername:port/controller\_if\_any}/Fetch) of your custom web service that contains your **Fetch** method.  
(http(s)://{servername:port/controller\_if\_any}/Fetch)

Click **Submit**.

## 3.1. Code

This section covers the details of the Authenticate and Fetch methods that must be implemented in your custom web service.


Your custom web service must include 2 methods Authenticate and Fetch. It can be written in any programming language not necessarily C# but C# will be used for the examples in this section.

(You can check the provided C# interface **IWebServiceLoginProvider**).

### 3.1.1. Fetch Method

The Fetch method is called when adding a new user in IAM using this login provider.

- From the Users menu, click on new user.
- Select you added web service provider.
- Fill the user's login code.

- Click on the validate button .

Your web service fetch method will be called.

```
public WebServiceUserModel Fetch(string loginCode);
```

A **loginCode** parameter will be sent to this method having as value the login code inserted when adding a new user.

In this method you can implement your needed criteria to find the user and return it.

This method should return **WebServiceUserModel**:

- FirstName: the user's first name.
- LastName: the user's last name.
- Email: the user's email.

### 3.1.2. Authenticate Method

The Authenticate method is called when an existing user (using this login provider) tries to login.

- From IAM login page, a user tries to login using his username (login code) and password
- The user clicks Login

Your web service authenticate method will be called.

```
public bool Authenticate(string username, string password);
```

The **username** (user's login code) and **password** provided by the user in the login page will be sent to this method and passed as parameters.

In this method you can implement your needed criteria to check if the username and password are correct and the user is authenticated.

This method should return a **bool** value specifying if the user is authenticated or not.

### 3.1.3. Example

The following is an example of a custom web service controller in C#:

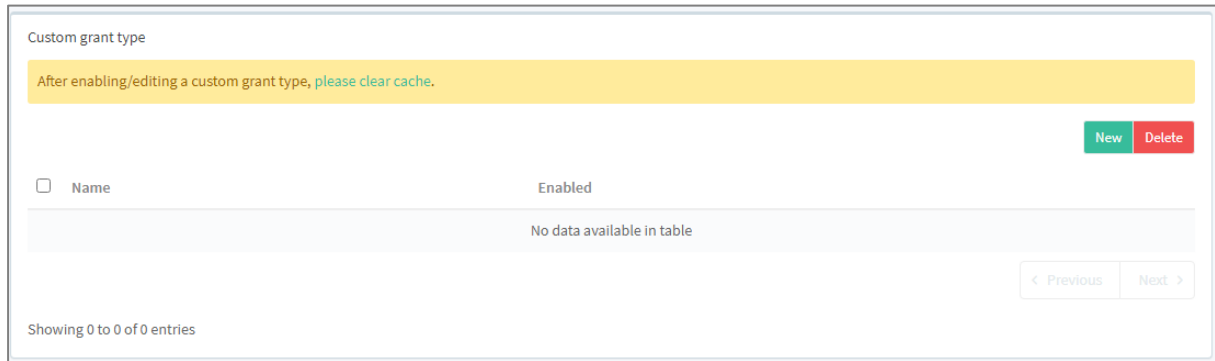
```
public class WebServiceController : ControllerBase
{
    [HttpGet]
    public IActionResult Fetch(string loginCode)
    {
        return Ok(new WebServiceUserModel { FirstName = "John", LastName = "Smith",
        Email = "john.smith@intalio.com" });
    }

    [HttpPost]
    public IActionResult Authenticate(string username, string password)
    {
        return Ok(true);
    }
}
```

## 4. Custom Grant Type

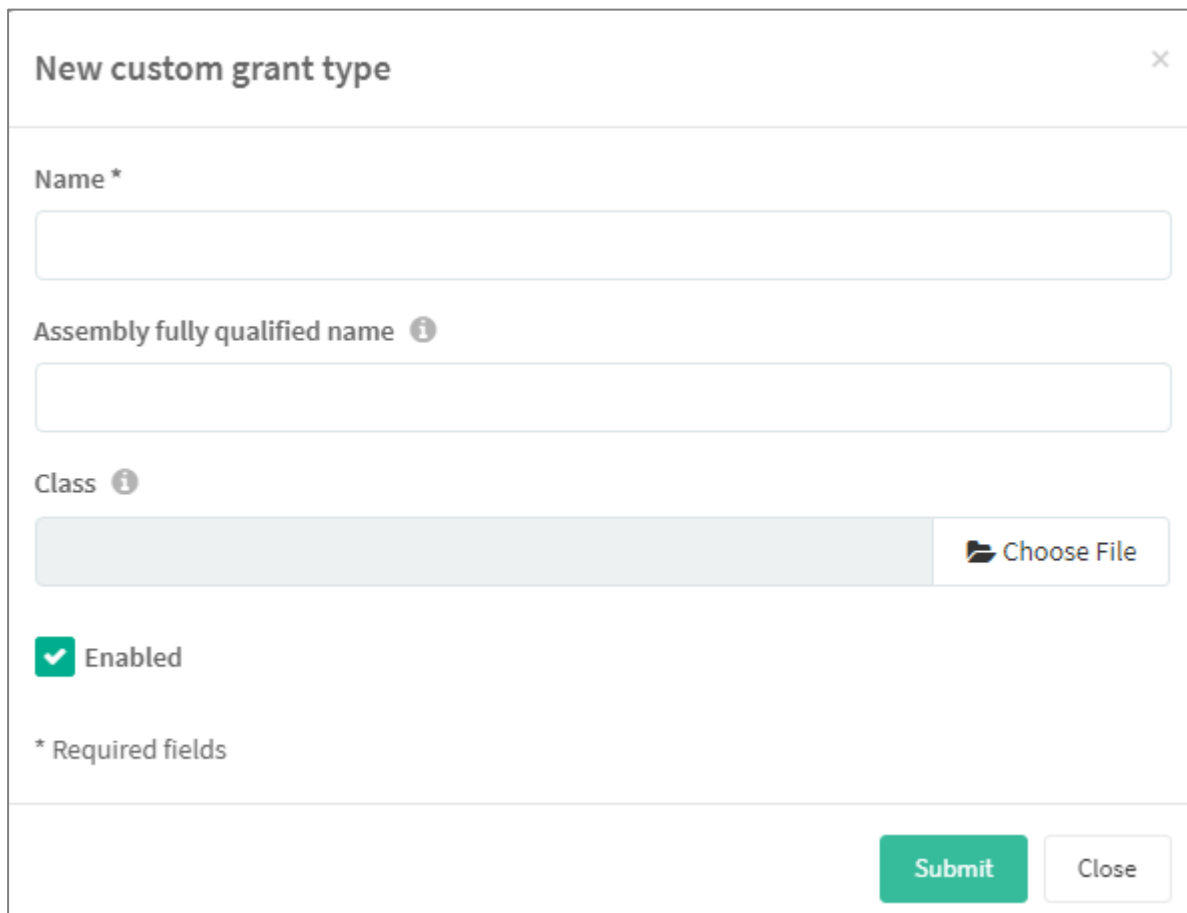
This section covers the details of custom grant type in login providers. Other login providers are not within the scope of this document, their details are covered in the admin guide.

The Custom grant types are available under the Login Providers menu.



The screenshot shows a web interface for managing custom grant types. At the top, there's a yellow banner with the text: "After enabling/editing a custom grant type, please clear cache." Below this, there are two buttons: "New" (green) and "Delete" (red). A table with columns "Name" and "Enabled" is shown, but it contains no data, with the message "No data available in table" in the center. At the bottom right of the table area, there are "Previous" and "Next" navigation buttons. At the bottom left, it says "Showing 0 to 0 of 0 entries".

To add a new grant type, click on the **new** button, the following form will open:



The screenshot shows a form titled "New custom grant type" with a close button (X) in the top right corner. The form contains the following fields and controls:

- Name \***: A text input field.
- Assembly fully qualified name ⓘ**: A text input field.
- Class ⓘ**: A text input field with a "Choose File" button (with a folder icon) to its right.
- Enabled**: A checkbox that is currently checked.
- \* Required fields**: A label indicating which fields are mandatory.
- Submit**: A green button at the bottom right.
- Close**: A button at the bottom right, next to the Submit button.

Fill the following:

- **Name:** grant type name.
- Either fill the assembly fully qualified name or upload a class.
  - **Class:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.ICustomGrantValidator`

- **Assembly fully qualified name:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.ICustomGrantValidator`
- **Enabled:** check this checkbox in case this grant type should be used.

Only **ONE** custom grant type can be enabled.

## Notes

- If the class uses any external library, the library DLL should be uploaded in IAM's **Assembly** section (check the admin guide)
- Assembly fully qualified name format:  
{namespace}.{classname}, {dll name}, Version={assembly version}, Culture=neutral, PublicKeyToken={null or the public key token if exist}
- Error that might occur:  
System.IO.FileLoadException: The given assembly name or codebase was invalid. (0x80131047)  
If this error occurred, please rewrite your assembly fully qualified name. This might occur if the value is copied and pasted and sometime on copy, characters like comma or space gets modified and will not be recognized when pasted.

## 4.1. Code

In this section the custom grant type class is explained. For details on how to create a solution and a class in .Net you can check the [custom solution](#) section.

**Note:** if your custom grant type did not work properly or an error occurred, you can check the IAM logs (Settings > logs) for errors.

Add new class implementing the interface **ICustomGrantValidator**.

When implementing the interface, 1 method will be created: *ValidateTokenAndGetUserIdAsync*.

```
public long? ValidateTokenAndGetUserIdAsync(string token);
```

A **token** parameter will be sent to this method having as value the external token sent as post request to /connect/token endpoint with grant\_type=custom.

In this method you can implement your needed criteria to get the user id from the claims in the external token.

This method should return a **long** value representing the IAM user id.



## 5. Two-Factor Authentication Provider

This section covers the details of two-factor authentication provider. If none is specified or none is set as default, the IAM 2FA provider will be used.

The Custom 2FA providers are available under the Two-factor authentication providers menu.

Two-factor authentication providers

New

After adding or setting a default two-factor authentication provider, the application must be restarted.

☐

Name

Default

Delete

No data available in table			
----------------------------	--	--	--

Showing 0 to 0 of 0 entries

< Previous

Next >

To add a new provider, click on the **new** button, the following form will open:

New two-factor authentication provider

Name \*

Assembly fully qualified name i

Class i

Choose File

☐ Use as default two-factor authentication provider

\* Required fields

Submit

Close



Fill the below:

- **Name:** two-factor authentication provider's name.
- Either fill the assembly fully qualified name or upload a class.
  - **Class:** class (.cs) implementing the interface `Intalio.IAM.Core.Interfaces.ITwoFactorAuthenticationProvider`
  - **Assembly fully qualified name:** the fully qualified name of a DLL (previously uploaded in the assembly section) containing a class implementing the interface `Intalio.IAM.Core.Interfaces.ITwoFactorAuthenticationProvider`
- **Use as default two-factor authentication provider:** check this checkbox to set this provider as the default. (Only one provider can be set as default)

Click **Submit**.

## 5.1. Code



This section covers the details of the `SendOTP` and `ValidateOTP` methods that must be implemented in your custom two-factor authentication provider.

Your custom provider must include 2 methods `SendOTP` and `ValidateOTP`. It can be written in any programming language not necessarily C# but C# will be used for the examples in this section.

(You can check the provided C# interface **`ITwoFactorAuthenticationProvider`**).

### 5.1.1. SendOTP

The `SendOTP` method is called in two cases:

1. When a user in IAM enable **Two-factor authentication** from his profile.
  - A user login to IAM.
  - Click on Profile  **Profile**.
  - Enable Two-factor authentication **Two-factor authentication** .
  - Click on **Submit**.

Your provider `SendOTP` method will be called.

2. When a user already has two-factor authentication enabled and is logging in to IAM
  - A user opens IAM and enters his username and password.

Your provider `SendOTP` method will be called.

```
public (bool, string) SendOTP(UserModel user);
```

A **user** parameter will be sent to this method having as value the user info with his attributes.

- **UserModel** containing the user's info:
  - **Id**: IAM's user id.
  - **FirstName**: first name.
  - **MiddleName**: middle name.
  - **LastName**: last name.
  - **FullName**: full name.
  - **Email**: email.
  - **Attributes**: the list of user attributes with their values related to the user.

In this method you can implement your own two-factor authentication.

This method should return **bool** and **string**:

- **bool**: true in case of success and false in case of error.
- **string**: null in case of success and the error message in case of failure.

The error message can be custom or can send one of the predefined messages from the enum **OTPMessage**.

## Note

In case of OTP by SMS, to get the user mobile number, a custom attribute should be added in IAM and it also should be added to the user attributes.

This attribute and its value will be sent to the SendOTP method in the attributes list of the UserModel object.

### 5.1.2. ValidateOTP

The ValidateOTP method is called when the user enters, on login or when enabling two-factor authentication, the OTP/verification code he received.

```
public (bool, string) ValidateOTP(UserModel user, string code);
```

Parameters passed to the method:

- **User**: UserModel containing the user's info with his attributes:
  - **Id**: IAM's user id.
  - **FirstName**: first name.
  - **MiddleName**: middle name.
  - **LastName**: last name.

- **FullName:** full name.
- **Email:** email.
- **Attributes:** the list of user attributes with their values related to the user.
- **Code:** the OTP/verification code the user has entered.

In this method you can implement your own criteria to validate the two-factor authentication code.

This method should return **bool** and **string**:

- bool: true in case of success and false in case of error.
- string: null in case of success and the error message in case of failure.

### 5.1.3. Example

The following is an example of a custom 2FA provider in C#.

In case of success:

```
public class TwoFACustom: ITwoFactorAuthenticationProvider
{
    public (bool, string) SendOTP(UserModel user)
    {
        //custom code implementing a custom 2FA provider
        return (true, null);
    }

    public (bool, string) ValidateOTP(UserModel user, string code)
    {
        //custom code to validate the OTP/verification code sent
        return (true, null);
    }
}
```

In case of error:

```
public class TwoFACustom: ITwoFactorAuthenticationProvider
{
    public (bool, string) SendOTP(UserModel user)
    {
        //custom code implementing a custom 2FA provider
        return (false, OTPMessage.OTPNotSent.ToString());
    }

    public (bool, string) ValidateOTP(UserModel user, string code)
    {
        //custom code to validate the OTP/verification code sent
        return (false, OTPMessage.InvalidOTP.ToString());
    }
}
```

## 6. Custom Solution

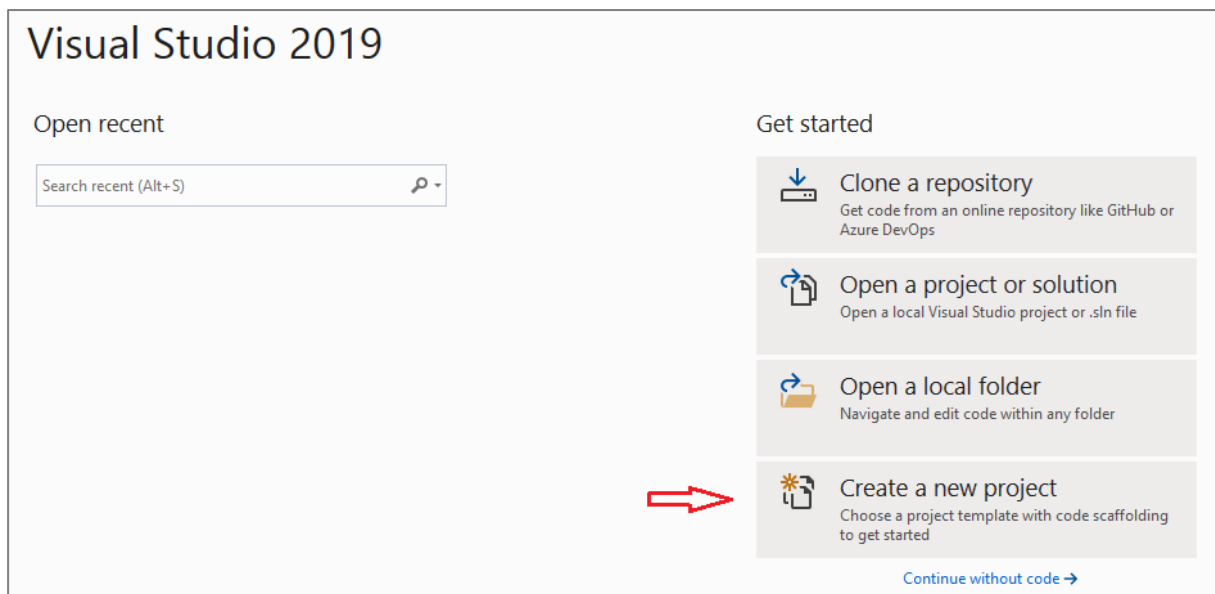
IAM is developed using ASP.Net core, the classes that you will create must be compatible with it. You can create a solution with ASP.Net core or .Net Standard 2.1.

Below are the steps to create custom classes using visual studio 2019 and .Net standard 2.1:

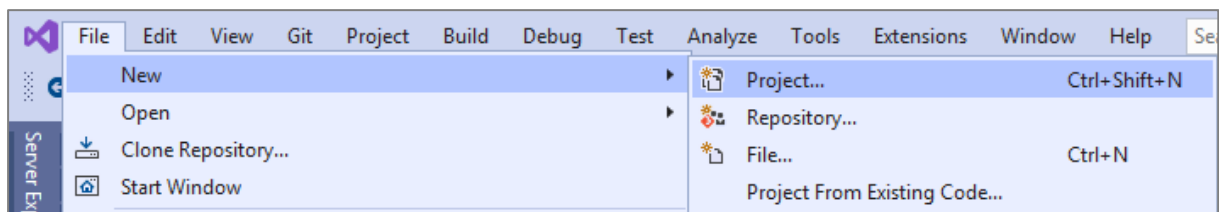
### Create a Solution

Open visual studio 2019.

From the home page, click on **Create a new project** or if you already have a solution opened, from the menu, click on **File > New > Project**.



Home Page – Create new project



Menu – Create new project






Select the project type **Class Library (.Net Standard)** and click next.

## Create a new project

Search for templates (Alt+S) Clear all

Recent project templates

C# All platforms All project types

-  **Console App (.NET Core)**  
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.  
C# Linux macOS Windows Console
-  **ASP.NET Core Web Application**  
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.  
C# Linux macOS Windows Cloud Service Web
-  **Blazor App**  
Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly (wasm). These templates can be used to build web apps with rich dynamic user interfaces (UIs).  
C# Linux macOS Windows Cloud Web
-  **Class Library (.NET Standard)**  
A project for creating a class library that targets .NET Standard.  
C# Android iOS Linux macOS Windows Library
-  **Azure Functions**  
A template to create an Azure Function project.

Back Next

Enter the project and solution names and select the location to save the project.

For this example, the project name will be **Intalio.IAM.Custom** and we are going to place the solution and project in the same directory. Then click **Create**.

## Configure your new project

Class Library (.NET Standard) C# Android iOS Linux macOS Windows Library

Project name

Intalio.IAM.Custom

Location

C:\Users\mal\Desktop\

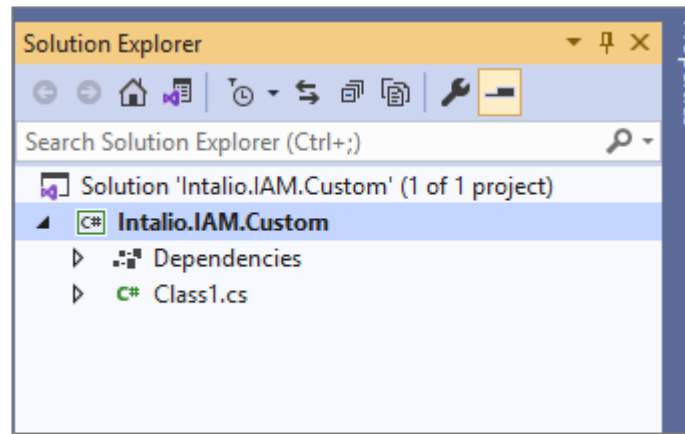
Solution name ⓘ

Intalio.IAM.Custom

☒ Place solution and project in the same directory

Back Create

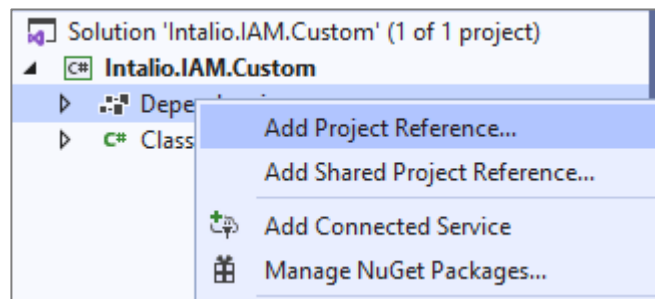
The following solution will be created.



## Add a Reference to Intalio.IAM.Core.dll

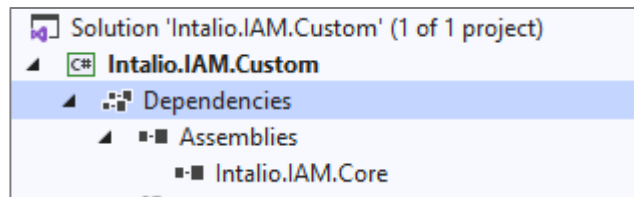
To add a reference to Intalio.IAM.Core.dll:

- Right click **Dependencies**.
- Click on **Add Project Reference**.



- Browse for the DLL **Intalio.IAM.Core.dll** located in your IAM package and click ok.

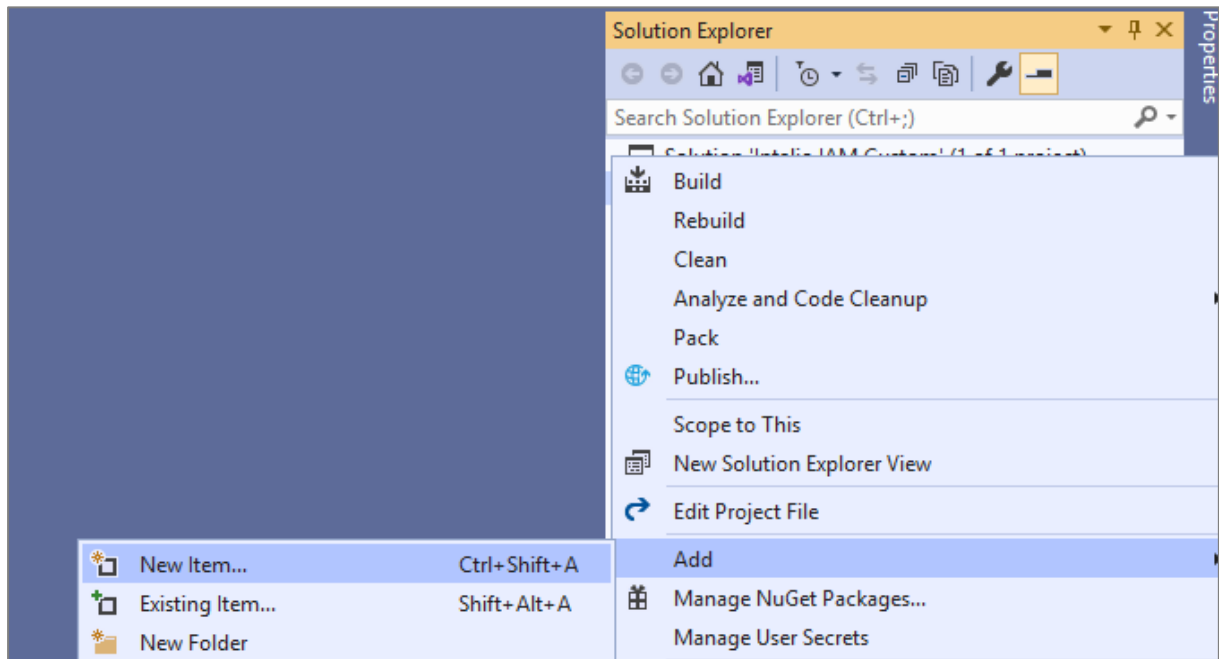
The DLL will be added to the solution.



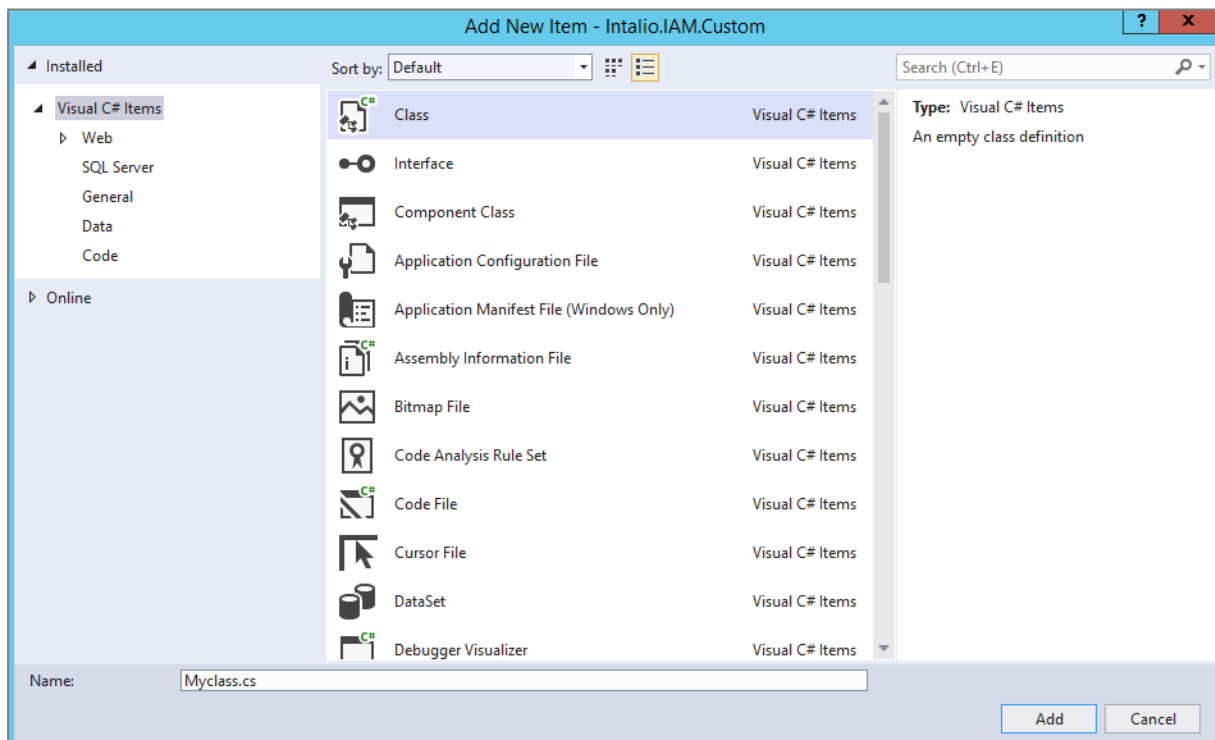
## Add New Class(es)

Add new class(es) implementing the related interface. Each is explained in its related section in this document.

To add a new class, right click on the project Intalio.IAM.Custom -> **Add** -> **New Item**.



Select **Class**, enter the class name and click **Add**.



## Final Step

Once finished and you have working class(es), you can:

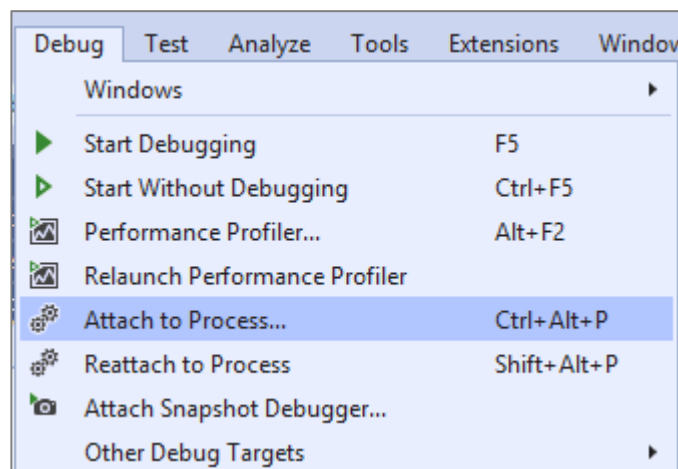
1. Upload the created classes in their related data source/validator
2. Or:
  - Build/publish your solution.
  - Upload the solution DLL in IAM's assembly section. (Check the admin guide)
  - Fill the **Assembly fully qualified name** in their related data source/validator

## Notes

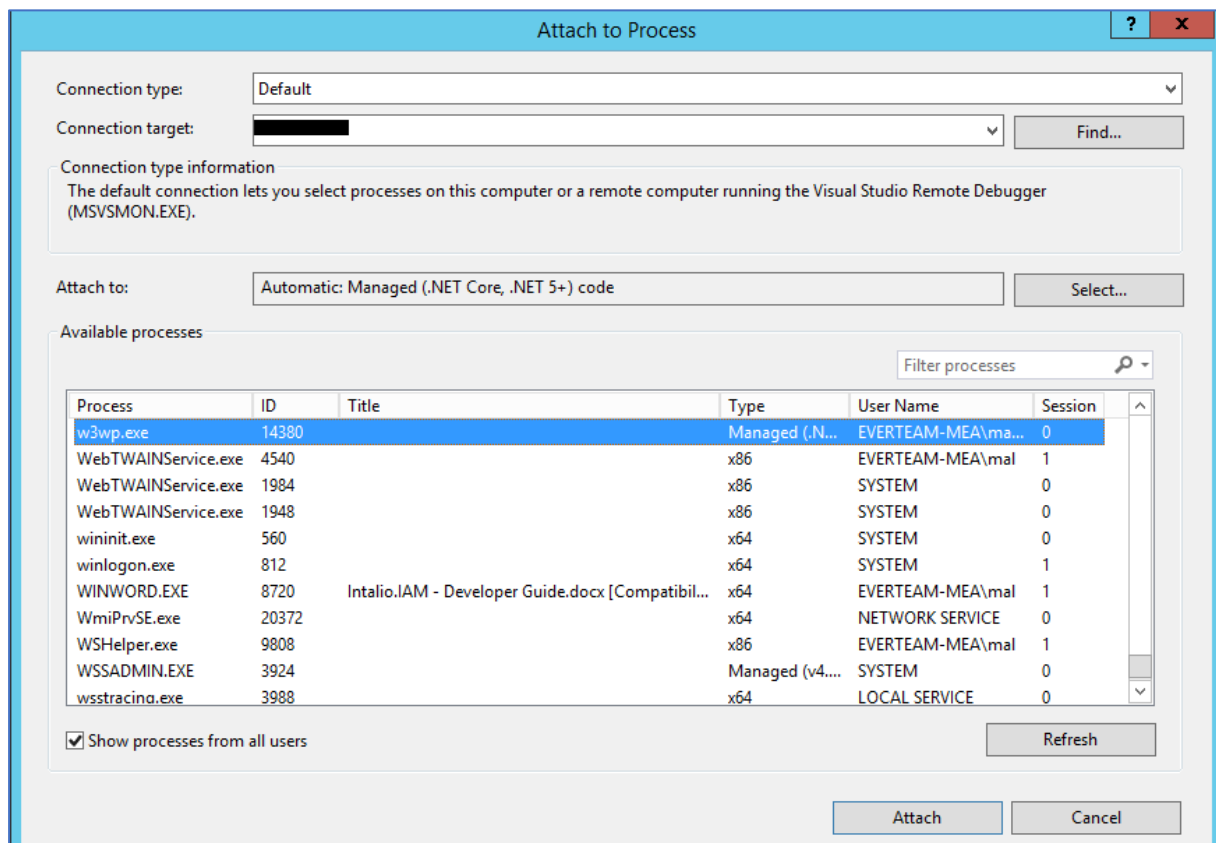
- Sample classes and jobs are provided with this document.
- If the class uses any external library, the library DLL should be uploaded in IAM's **Assembly** section.

## Hints

- You can debug your code by using the assembly fully qualified name instead of uploading a class. Make sure you have the latest DLL uploaded in IAM.
  - Upload your DLL in the assembly section (if a DLL is previously uploaded, delete it then re-upload the latest)
  - From visual studio, set your breakpoints then click on **Debug > Attach to process** and select the **w3wp.exe** process and click **Attach**.







- Execute your job from IAM.

If the debugging did not work, perform IISRESET, refresh IAM, then recheck the previous steps.

- You can put logs and messages in your code and show them in IAM logs section under the Settings menu by simply adding this line in your code:

```
ExceptionLogger.LogException("your message");
```

*Just don't forget to remove them once done and you have a working class.*