Step by Step Guide: IdentityIQ Plugin development

Author: **Kalyan Kumar Saha**

DISCLAIMER: *This document does not intend to replace SailPoint published knowledge resources available in Compass community. This document is solely for giving a direction to beginners to start exploring plugin based on my experience and learning.*

Overview

IdentityIQ plugin is one of the most powerful features that provide immense freedom in implementing "*Non-standard IAM requirements*" to be embedded within IIQ. "*Non-Standard IAM requirements*" term refers to those use cases which are independent but do not typically relate to identity management.

For example, setting up an interface to keep a track of daily IIQ login sessions (who accesses IIQ when). This requirement can typically be resolved by using an access manager, but plugin allows developing a solution within IIQ if access manager route is not an option.

Knowledge Resources

Plugin development bible is here. The hyperlink of each component of plugin is well described here.

In this article, we shall understand each step with example i.e. **"Contractor Management System"** so that plugin development techniques become crystal clear.

"Contractor Management System" (CMS) is aimed to automate onboarding contractors into IIQ. The plugin presents an interactive UI to enter contractor details, validates user inputs against backend store to ensure data integrity and then submits an identity creation request to SailPoint.

Knowledge Pre-requisite

- Fair knowledge on RDBMS (DDL, DML etc.)
- Good knowledge in Java Programming and JDBC
- Good knowledge around web development i.e. HTML, JavaScript, CSS
- Good knowledge around IIQ basics and APIs

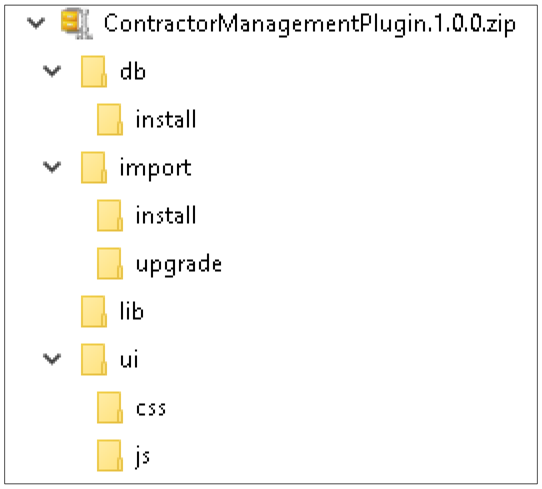
[**Note**: **AngularJS** is apt for IIQ plugins as it is recommended for Single Page Applications (SPA). Most of the SailPoint published IIQ plugins leverage AngularJS. So, it is good time to learn it from here.]

Test Bed Details

- Apache Tomcat 7.0
- MySQL Database 5.6
- SailPoint IdentityIQ 7.2
- XHTML, Angular JS, Java 1.8

Plugin Package Structure

Any plugin needs to be packaged in zipped format (.zip) containing below folders:



- **db** - This folder contains the DDL scripts to create database objects to persist your plugin data. IIQ runs the script only once during plugin installation and creates the DB objects like tables, indexes, constraints etc. under '**identityiqplugin**' database. The script needs to reside within '**install**' subfolder and the file name MUST be ***install.<dbvendor>*** where *<dbvendor>* is one of following based on environment: *mysql, db2, oracle, sqlserver*

- **import** – This folder contains the SailPoint object XMLs to be imported e.g. Configuration, QuickLink, SPRight, Capability, Rule, TaskDefinition etc. Objects present under '**install**' subfolder will be imported into IIQ during plugin installation and further update of the same plugin will import objects from '**upgrade**' subfolder. So, it is very important that both subfolders contain same set of XMLs.

- **lib** – This folder contains jar files bundling compiled Java classes for plugin REST services, task/service executors, any third-party Java library etc.

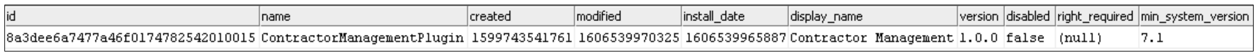
- **ui** – This folder contains at least one xhtml file with name '***page.xhtml***' mandatorily. The XHTML file is launched to render UI when a plugin is accessed. Modular approaches can be adopted to keep UI elements separated into multiple xhtml files based on requirement. **'css', 'js'** subfolders are generally used to keep any CSS file and Javascript libraries respectively though it is not mandatory to have those folders but good to follow this practice.

- **manifest.xml** – This is plugin definition XML file containing version, REST service class declaration, executors, runtime settings etc. Refer attached CMS plugin manifest.xml file.

IIQ Database Table Reference

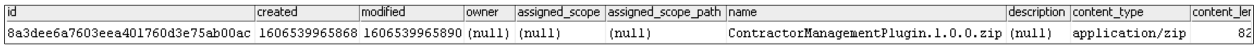
Three IIQ DB tables are involved in persisting plugin configuration details:

1. When a plugin is installed, configuration details are saved in **identityiq.spt_plugin** table.



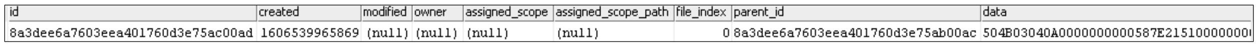
id	name	created	modified	install_date	display_name	version	disabled	right_required	min_system_version
8a3dee6a7477a46f0174782542010015	ContractorManagementPlugin	1599743541761	1606539970325	1606539965887	Contractor Management	1.0.0	false	(null)	7.1

2. **identityiq.spt_persisted_file** table contains the plugin zip file related metadata



id	created	modified	owner	assigned_scope	assigned_scope_path	name	description	content_type	content_le
8a3dee6a7603eea401760d3e75ab00ac	1606539965868	1606539965890	(null)	(null)	(null)	ContractorManagementPlugin.1.0.0.zip	(null)	application/zip	82

3. Plugin zip file content is saved in **identityiq.spt_file_bucket** table as BLOB under 'data' column. IIQ refers to the BLOB during runtime to access plugin files like page.xhtml, load plugin Java classes without needing to restart IIQ web container (e.g. Tomcat).



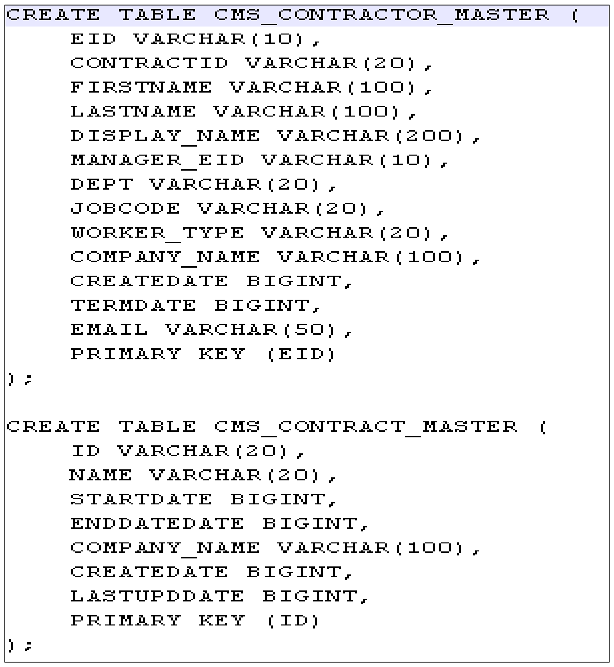
id	created	modified	owner	assigned_scope	assigned_scope_path	file_index	parent_id	data
8a3dee6a7603eea401760d3e75ac00ad	1606539965869	(null)	(null)	(null)	(null)	0	8a3dee6a7603eea401760d3e75ab00ac	504B03040A0000000000587E2151000000

Steps towards First Successful Plugin

- Learn a JavaScript framework like JQuery or AngularJS. It helps to design interactive UI along with accomplish REST calls to plugin webservices.

- Create a .sql file under '***db\install***' folder and put all the DDL statements to create DB objects.

  CMS plugin needs three tables to implement the functionality. So, *install.mysql* contains necessary CREATE TABLE statements.



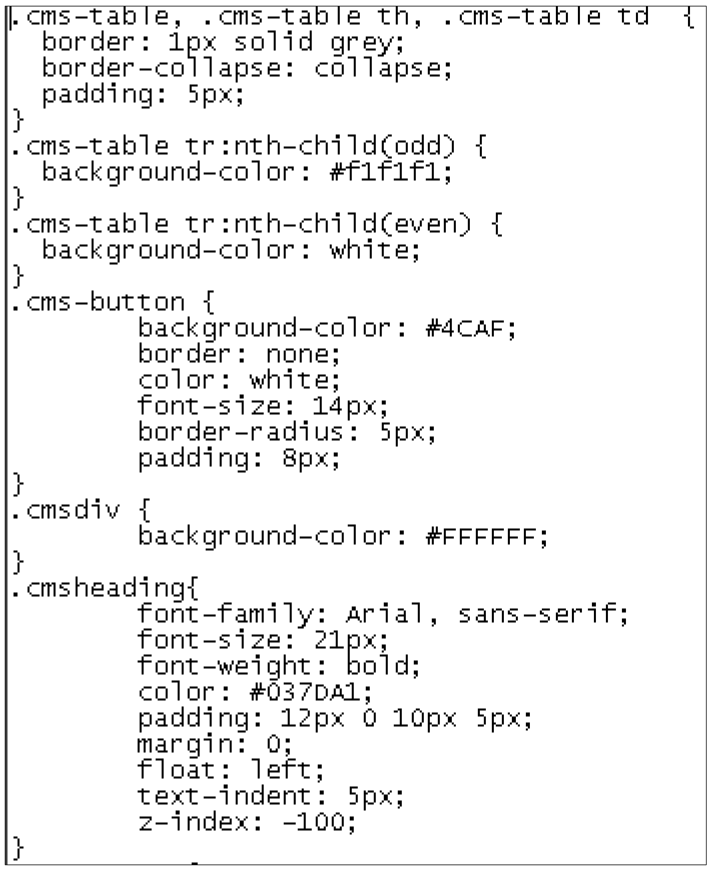
```
CREATE TABLE CMS_CONTRACTOR_MASTER (
     EID VARCHAR(10),
     CONTRACTID VARCHAR(20),
     FIRSTNAME VARCHAR(100),
     LASTNAME VARCHAR(100),
     DISPLAY_NAME VARCHAR(200),
     MANAGER_EID VARCHAR(10),
     DEPT VARCHAR(20),
     JOBCODE VARCHAR(20),
     WORKER_TYPE VARCHAR(20),
     COMPANY_NAME VARCHAR(100),
     CREATEDATE BIGINT,
     TERMDATE BIGINT,
     EMAIL VARCHAR(50),
     PRIMARY KEY (EID)
);

CREATE TABLE CMS_CONTRACT_MASTER (
     ID VARCHAR(20),
     NAME VARCHAR(20),
     STARTDATE BIGINT,
     ENDDATEDATE BIGINT,
     COMPANY_NAME VARCHAR(100),
     CREATEDATE BIGINT,
     LASTUPDDATE BIGINT,
     PRIMARY KEY (ID)
);
```

- Start UI development by creating a file *page.xhtml* under '***ui***' folder and put UI html elements (text boxes, text fields, buttons checkboxes etc.) in it. This must accompany the JavaScript code to allow data flow from view (UI) to model (Java backend) and vice versa by means of REST webservices. It is better to keep JavaScript code separate in a .js file instead of squeezing it within <script> tag of the html file.

- If CSS is used to make UI attractive, then make sure no CSS selector name collides with IdentityIQ CSS selectors as it would screw up existing IIQ pages if any conflict occurs.

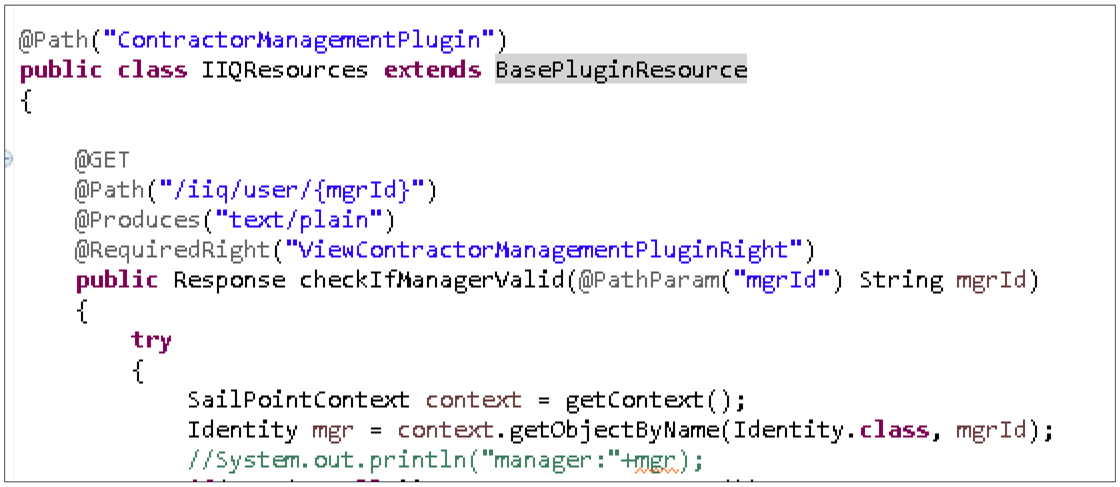
  In CMS plugin, you will see the selector names are prefixed with *cms* to make it unique.



```
.cms-table, .cms-table th, .cms-table td  {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
.cms-table tr:nth-child(odd) {
  background-color: #f1f1f1;
}
.cms-table tr:nth-child(even) {
  background-color: white;
}
.cms-button {
        background-color: #4CAF;
        border: none;
        color: white;
        font-size: 14px;
        border-radius: 5px;
        padding: 8px;
}
.cmsdiv {
        background-color: #FFFFFF;
}
.cmsheading{
        font-family: Arial, sans-serif;
        font-size: 21px;
        font-weight: bold;
        color: #037DA1;
        padding: 12px 0 10px 5px;
        margin: 0;
        float: left;
        text-indent: 5px;
        z-index: -100;
}
```

- Create IIQ objects like QuickLink, SPRight, Capability, Workflow, and TaskDefinition etc. to support plugin activity. Refer to attached CMS plugin to understand more on this

- Start writing REST plugin web services code once basic UI is ready. From this part, Java development work starts. An IDE (e.g. Eclipse) can be used to complete Java coding
    - Plugin front-end interacts with IIQ using REST webservice. User inputs are passed to the webservice as part HTTP request parameter in case of GET or request body in case of POST for processing.
    - Feel free to create your own package structure. A Java class needs to extend *sailpoint.rest.plugin.BasePluginResource* to create webservice backend.
    - Annotations are used to create service endpoints and method to perform business logic. For CMS plugin, two REST resources are created: one to perform insert/update/query to CMS plugin tables and another one to interact with IIQ to validate user entered contractor data and create contractor identity cube.

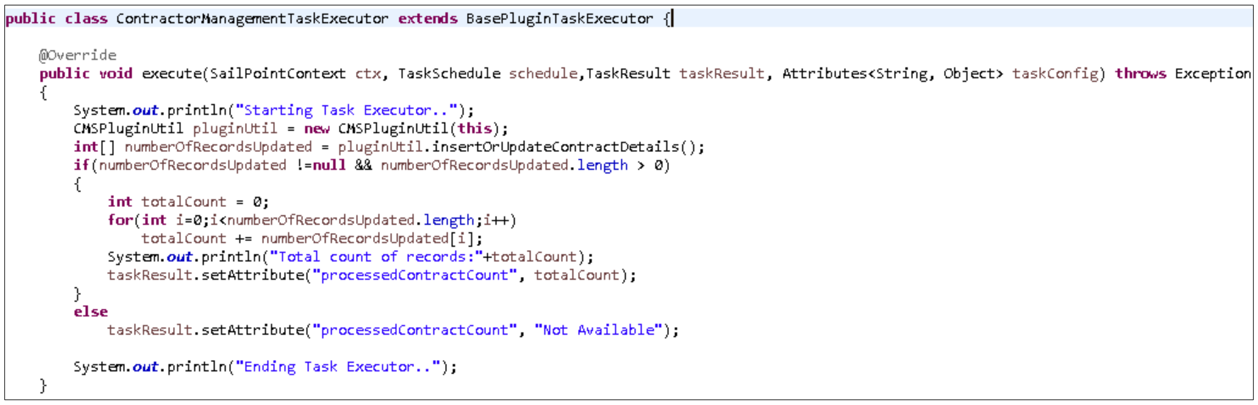


```java
@Path("ContractorManagementPlugin")
public class IIQResources extends BasePluginResource
{

    @GET
    @Path("/iiq/user/{mgrId}")
    @Produces("text/plain")
    @RequiredRight("ViewContractorManagementPluginRight")
    public Response checkIfManagerValid(@PathParam("mgrId") String mgrId)
    {
        try
        {
            SailPointContext context = getContext();
            Identity mgr = context.getObjectByName(Identity.class, mgrId);
            //System.out.println("manager:"+mgr);
```

   - There are a few built in methods available to get plugin DB connection, SailPoint contexts etc. which make coding easy

- If plugin task executor is needed, then a Java class needs to extend *sailpoint.task.BasePluginTaskExecutor* interface and implement execute() method. Task executor is needed when any task is required to perform any bulk work in background to support plugin functionality.
  In CMS plugin, a task executor is created to synchronize contract data from an external DB table *contractdb.contract_master* into plugin DB table *cms_contract_master*. CMS plugin later uses these data to validate contractor onboarding request.

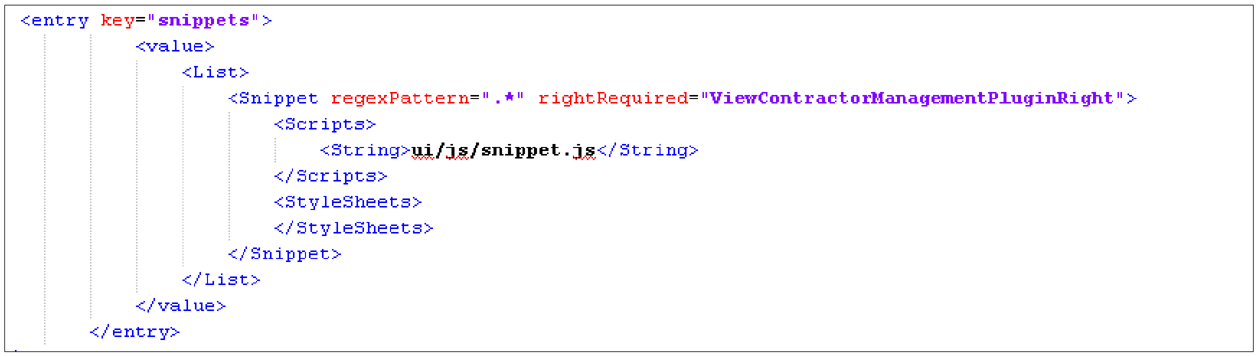


```java
public class ContractorManagementTaskExecutor extends BasePluginTaskExecutor {

    @Override
    public void execute(SailPointContext ctx, TaskSchedule schedule,TaskResult taskResult, Attributes<String, Object> taskConfig) throws Exception
    {
        System.out.println("Starting Task Executor..");
        CMSPluginUtil pluginUtil = new CMSPluginUtil(this);
        int[] numberOfRecordsUpdated = pluginUtil.insertOrUpdateContractDetails();
        if(numberOfRecordsUpdated !=null && numberOfRecordsUpdated.length > 0)
        {
            int totalCount = 0;
            for(int i=0;i<numberOfRecordsUpdated.length;i++)
                totalCount += numberOfRecordsUpdated[i];
            System.out.println("Total count of records:"+totalCount);
            taskResult.setAttribute("processedContractCount", totalCount);
        }
        else
            taskResult.setAttribute("processedContractCount", "Not Available");

        System.out.println("Ending Task Executor..");
    }
```

- All Java classes would need to be bundled in a jar and keep that jar in ‘lib’ folder of plugin package
- Create *manifest.xml* and put configuration details correctly
- Once all files are ready, create a .zip with the above-mentioned folders to get ready for plugin installation
- While development/troubleshooting is in progress, above steps are repetitive until full requirement is completely and correctly implemented. Either a build script can be developed (discussed here) or use software (e.g. WinZip) to create plugin zip manually every time before installing the plugin
- If JavaScript injection is to be used to create menu option, a snippet file should also be kept under ‘***ui\js***’ folder and the same needs to be referred in manifest.xml file under <Snippet> tag

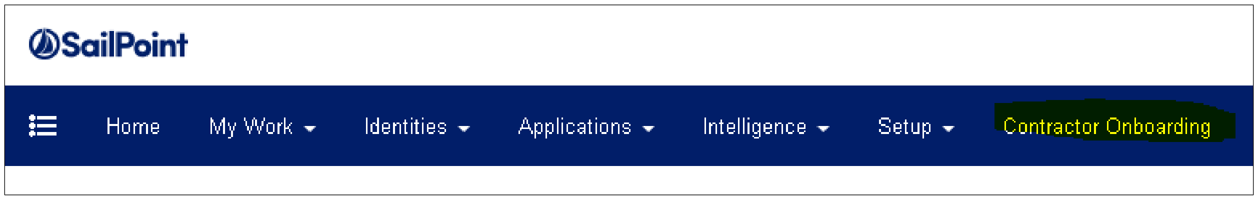


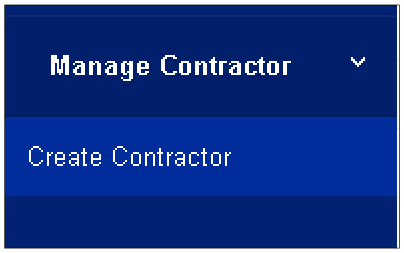
```
<entry key="snippets">
        <value>
            <List>
                <Snippet regexPattern=".*" rightRequired="ViewContractorManagementPluginRight">
                    <Scripts>
                        <String>ui/js/snippet.js</String>
                    </Scripts>
                    <StyleSheets>
                    </StyleSheets>
                </Snippet>
            </List>
        </value>
    </entry>
```

Screenshots

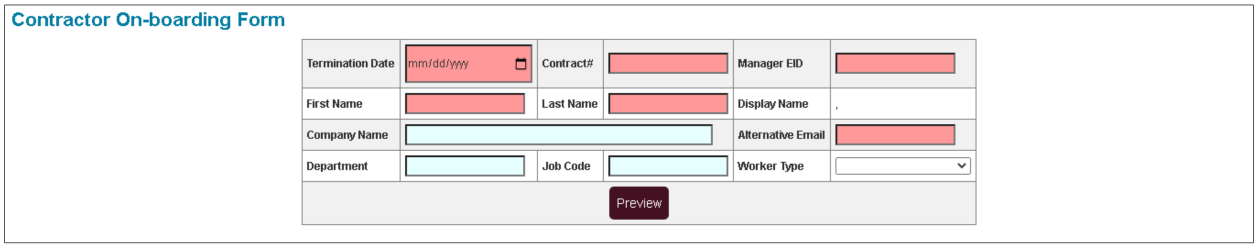
There are two ways to launch this plugin: a. A menu item created through JavaScript Injection or b. quicklink. Note here, both quicklink or menu item will be visible only those users having '*ViewContractorManagementPluginRight*' or System administrators.

Both basically invokes plugin through an URL http://<host>:<port>/identityiq/plugins/pluginPage.jsf?pn=<pluginname>> where <pluginname> is the name attribute value in manifest.xml file





The plugin launches page.xhtml to render UI



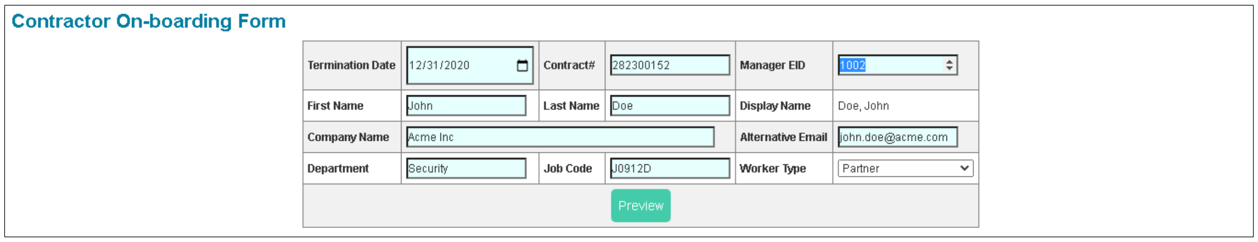
Contractor On-boarding Form

Termination Date	mm/dd/yyyy	Contract#		Manager EID	
First Name		Last Name		Display Name	
Company Name				Alternative Email	
Department		Job Code		Worker Type	

Preview

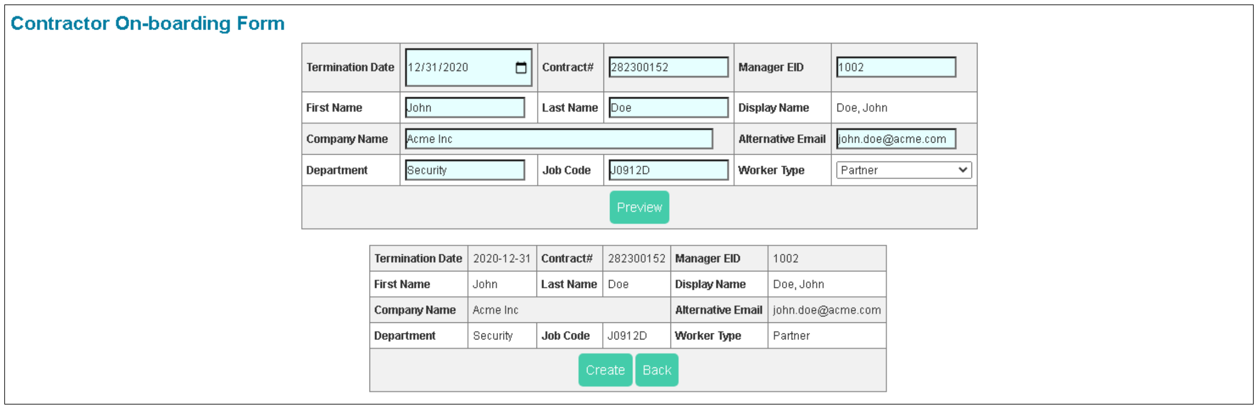
Red colored fields are mandatory fields. Until all mandatory fields are provided with correct values, '*Preview'* button will not be clickable. The form validates '*Contract#*' field against a CMS table data i.e. *cms_contract_master* and '*Manager EID'* field against a valid manager cube present in IIQ through plugin REST call.

Once all data are entered, 'Preview' button will be enabled and a read-only form will be shown up to verify the details upon clicking on the button.



Contractor On-boarding Form

Termination Date	12/31/2020	Contract#	282300152	Manager EID	1002
First Name	John	Last Name	Doe	Display Name	Doe, John
Company Name	Acme Inc			Alternative Email	john.doe@acme.com
Department	Security	Job Code	J0912D	Worker Type	Partner

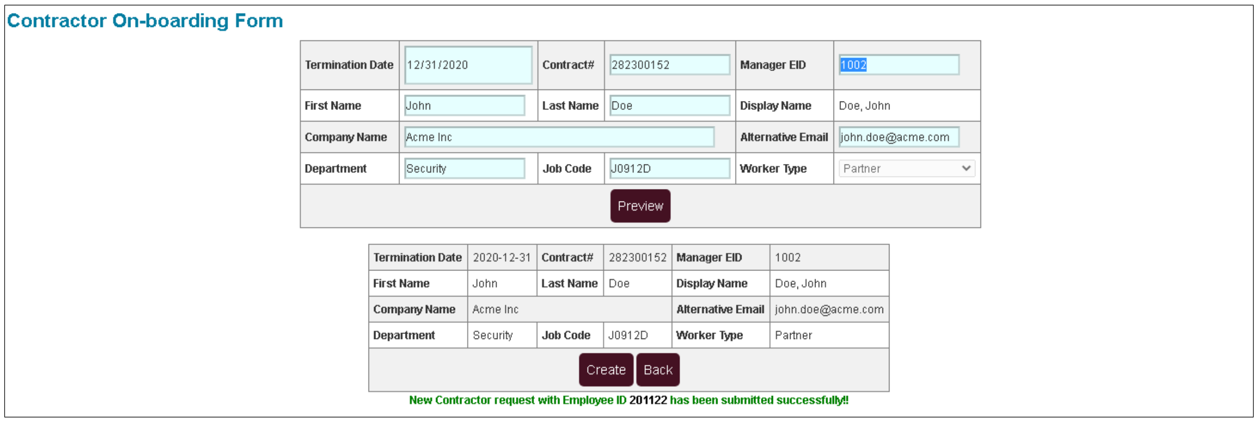


Contractor On-boarding Form

Termination Date	12/31/2020	Contract#	282300152	Manager EID	1002
First Name	John	Last Name	Doe	Display Name	Doe, John
Company Name	Acme Inc			Alternative Email	john.doe@acme.com
Department	Security	Job Code	J0912D	Worker Type	Partner

Termination Date	2020-12-31	Contract#	282300152	Manager EID	1002
First Name	John	Last Name	Doe	Display Name	Doe, John
Company Name	Acme Inc			Alternative Email	john.doe@acme.com
Department	Security	Job Code	J0912D	Worker Type	Partner

User can go back and re-enter details if needed or click on '*Create*' button. Upon clicking on the button, all buttons and inputs fields will be disabled to prevent any further change. It will create contractor record in CMS table *cms_contractor_master* as well an identity cube in IIQ. The employee ID is generated dynamically, and the success message shows the allocated employee ID.

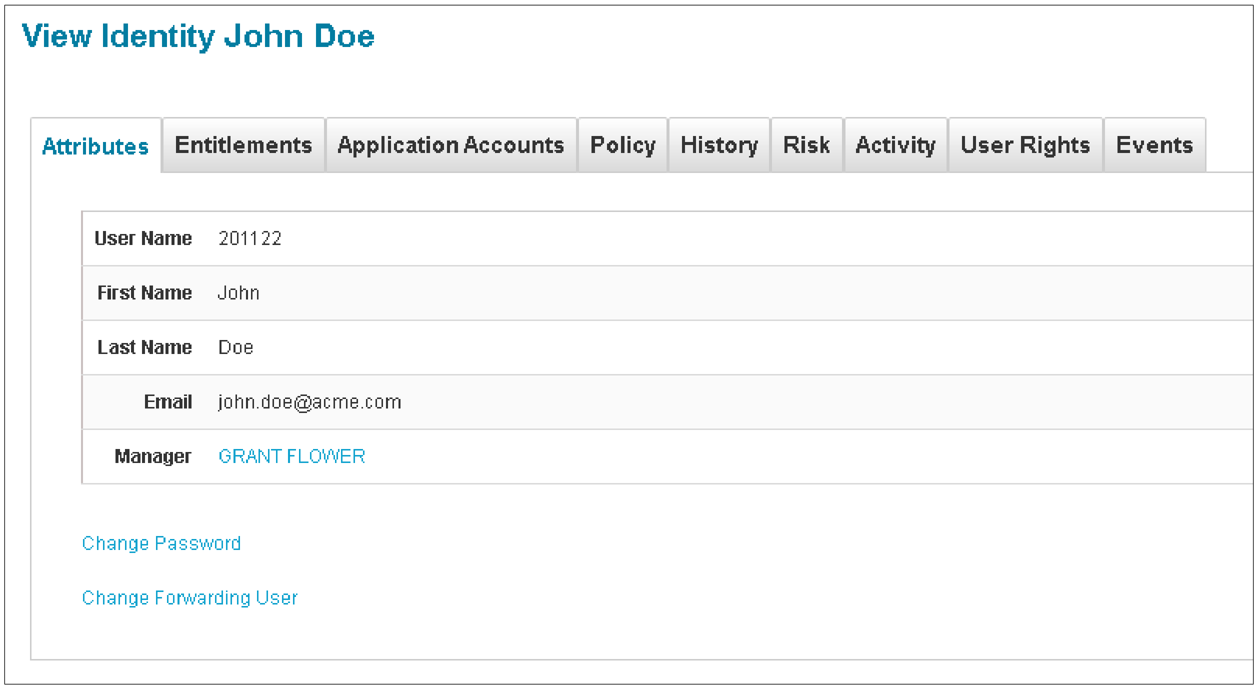


Contractor On-boarding Form

Termination Date	12/31/2020	Contract#	282300152	Manager EID	1002
First Name	John	Last Name	Doe	Display Name	Doe, John
Company Name	Acme Inc			Alternative Email	john.doe@acme.com
Department	Security	Job Code	J0912D	Worker Type	Partner

Termination Date	2020-12-31	Contract#	282300152	Manager EID	1002
First Name	John	Last Name	Doe	Display Name	Doe, John
Company Name	Acme Inc			Alternative Email	john.doe@acme.com
Department	Security	Job Code	J0912D	Worker Type	Partner

New Contractor request with Employee ID **201122** has been submitted successfully!!

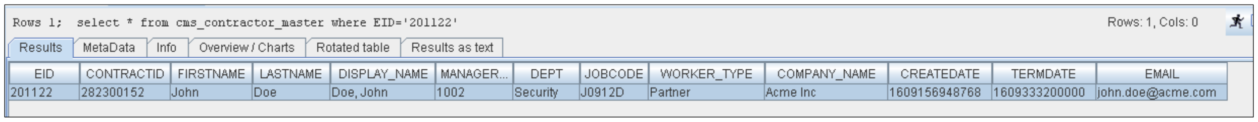


View Identity John Doe

User Name	201122
First Name	John
Last Name	Doe
Email	john.doe@acme.com
Manager	GRANT FLOWER

Change Password

Change Forwarding User

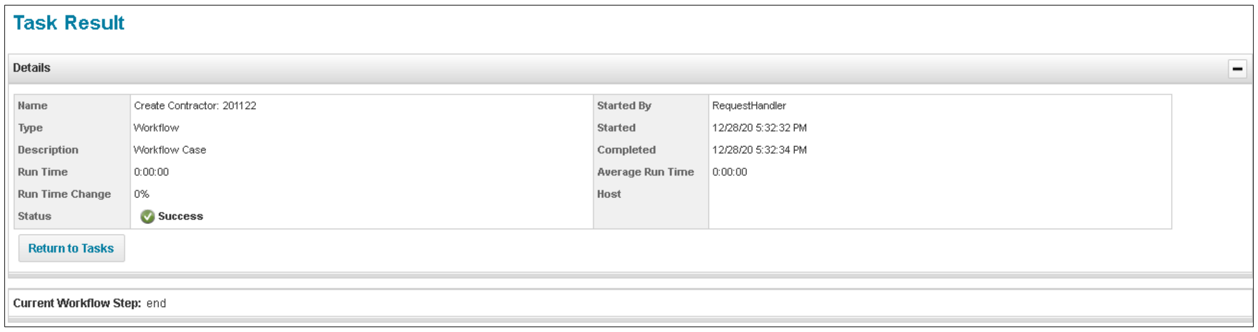


```
Rows 1;  select * from cms_contractor_master where EID='201122'
```

Rows: 1, Cols: 0

EID	CONTRACTID	FIRSTNAME	LASTNAME	DISPLAY_NAME	MANAGER...	DEPT	JOBCODE	WORKER_TYPE	COMPANY_NAME	CREATEDATE	TERMDATE	EMAIL
201122	282300152	John	Doe	Doe, John	1002	Security	J0912D	Partner	Acme Inc	1609156948768	1609333200000	john.doe@acme.com

A task result is also created to track the progress of this contractor creation in IIQ.



Task Result

Details

Name	Create Contractor: 201122	Started By	RequestHandler
Type	Workflow	Started	12/28/20 5:32:32 PM
Description	Workflow Case	Completed	12/28/20 5:32:34 PM
Run Time	0:00:00	Average Run Time	0:00:00
Run Time Change	0%	Host	
Status	✓ Success		

Current Workflow Step: end

Plugin Artifacts

ContractorManagementPlugin.1.0.0.zip