IM473673

Shorten Lead Times by Integrating Design Automation in Downstream Processes

Peter Van Avondt
Autodesk

Sven Dickmans
Autodesk

Learning Objectives

- Learn about the possibilities of the Forge Design Automation API for Inventor.
- Discover business-process capabilities of Fusion Lifecycle.
- Discover the steps it takes to make an integration between Design Automation for Inventor and Fusion Lifecycle.
- Discover a sample integration between Design Automation for Inventor and Fusion Lifecycle.

Description

More and more companies are moving to a configure-to-order approach, making their products modular and configurable. Where in the past each customer order was designed separately, with a product configurator, every order is compiled from existing modules. The automated output of the configured product (quote, visuals, Bill of Materials) can be used in the downstream processes. This means a significant reduction in the turnaround time for lengthy processes such as quoting, planning, and engineering. The Forge platform offers with the Design Automation API for Inventor software a technology to build a web-based configurator, automating BOM, pricing, and 3D-visualization delivery. Besides that, Fusion Lifecycle software offers the perfect platform for managing the business-wide workflows, like the management of downstream engineering, sales, and change processes. In this class, we will show you an example of how to build and combine these technologies to get an integrated process.

Speaker(s)

Peter Van Avondt | Technical Sales Specialist – Autodesk EMEA

Peter Van Avondt works for Autodesk as a Technical Specialist Data Management PDM/PLM in Northern Europe, based in Belgium. After graduating as a master in electromechanical engineering he joined an Autodesk channel partner as a technical consultant specialized in 3D CAD and product data management PDM. For the last 17 years he has built up a lot of experience in variety of Autodesk design tools as well as with Autodesk Vault and Fusion Lifecycle. In his current role he uses this wealthy knowledge helping Autodesk resellers, prospects, and customers to adopt and implement the Autodesk solutions across different industries including industrial machinery, pharmaceutical, architecture, engineering and construction companies.

Sven Dickmans | Technical Sales Specialist – Autodesk EMEA

Sven helps customers, prospects, and partners in achieving excellence of business process execution with cloud based PDM/PLM solutions from Autodesk. He also engages in developing new collaboration solutions using connected cloud services of Forge. Sven is part of Autodesk's technical specialists' team in Germany.

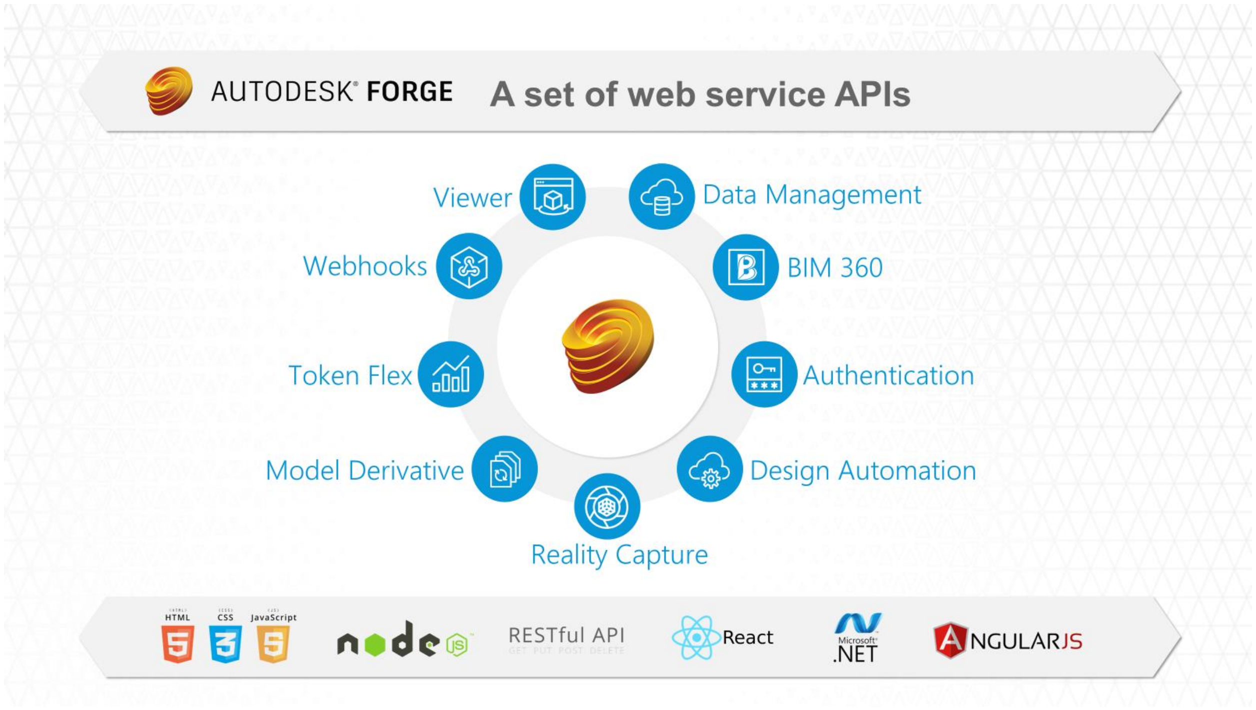
Contents

Cloud based automation with Fusion Lifecycle & Forge

Cloud Based Automation Services

With **Autodesk Forge** you can build online workflows and experiences around your design data, create, modify or read your data in web applications and create workflows linking Autodesk web applications to other enterprise applications.

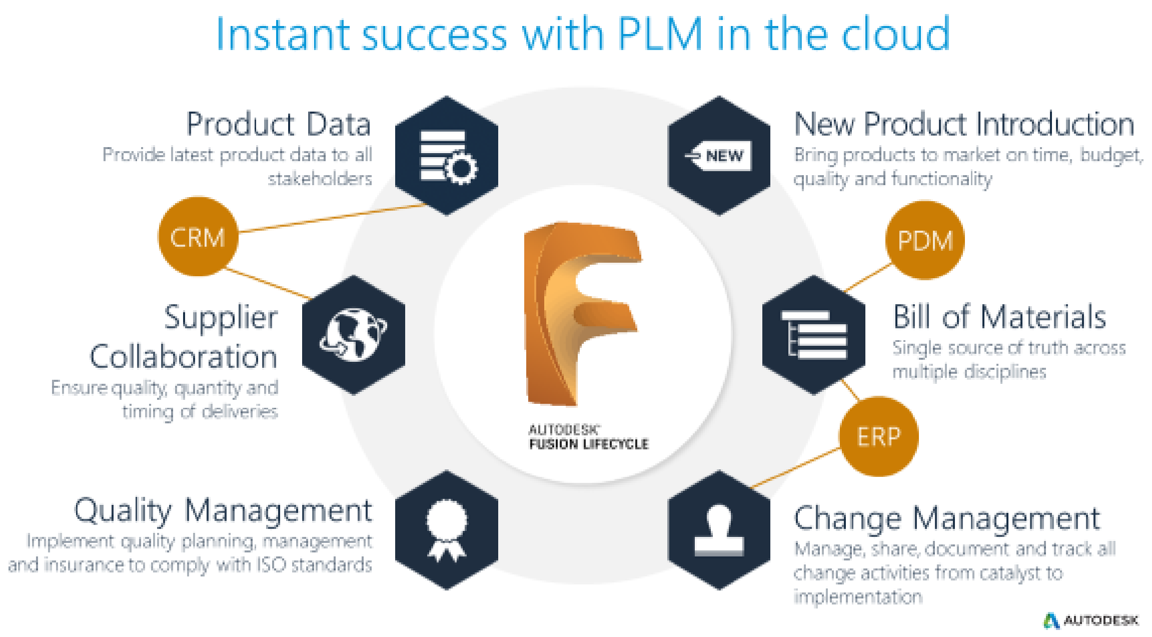
Autodesk Forge consist of Components of the technology stack that Autodesk uses to build its own web applications.



Cloud data & process management

Autodesk **Fusion Lifecycle** is a common platform for data, processes and deliverables across all business units. It is accessible to everyone, easy to use to meet various stakeholders needs and flexible to be adopted to customer needs.

Fusion Lifecycle captures any kind of data in a smart manner. You can upload any kind of information and store in a smart form. There are preconfigured forms for Change Requests, Change Orders, Problem Reports, Projects, Tasks and so on. They can be easily configured to the company’s needs. These forms provide capabilities to link other records and to apply automatic validations (like mandatory fields). This makes the information smart for easy reuse. Users can report on this information and search or filter for any detail. Colour indicators even may be used to highlight information of special relevance.



Instant success with PLM in the cloud
Product Data
Provide latest product data to all stakeholders
New Product Introduction
Bring products to market on time, budget, quality and functionality
CRM
PDM
Supplier Collaboration
Ensure quality, quantity and timing of deliveries
Bill of Materials
Single source of truth across multiple disciplines
NEW
AUTODESK FUSION LIFECYCLE
ERP
Quality Management
Implement quality planning, management and insurance to comply with ISO standards
Change Management
Manage, share, document and track all change activities from catalyst to implementation
AUTODESK

The key main business processes are shown in this graphic.

The data is also connected to online processes and workflows. These workflows automatically pass related information to all the stakeholders involved. They also may provide alternate next steps to drive and support decisions in the business process flows. Processes can also be preconfigured and can be adjusted by the customers individually.

Processes and projects require people's involvement. Tasks can be defined easily by reusing templates to request these deliverables from anyone along the product lifecycle. Due dates and priorities can be used to let assignees easily sort their personal to-do list by the given criteria. The assignees will be informed about relevant evens automatically by mail, e.g. when a task gets assigned or when it becomes due.

Combining the power of both

In this class we combine the power of both cloud platforms to showcase a workflow where we manage Product Data and the introduction in the market on one hand and configurations and RFQs in the other hand.

ENDLESS POSSIBILITIES
CLOUD DESIGN AUTOMATION SERVICES
CLOUD DATA & PROCESS MANAGEMENT

The use cases that we have presented in our solution are:

- **Product Catalog**: Browse product portfolio being managed in Fusion Lifecycle and retrieve product details
- **Online Product Configurator**: Derive product configuration of interest in Product Catalog by using iLogic rules
- **Variant Management:** Storage of derived configurations for reuse and statistical analysis
- **RFQ Processes:** Online processing of Requests for quote with seamless handover of product configuration

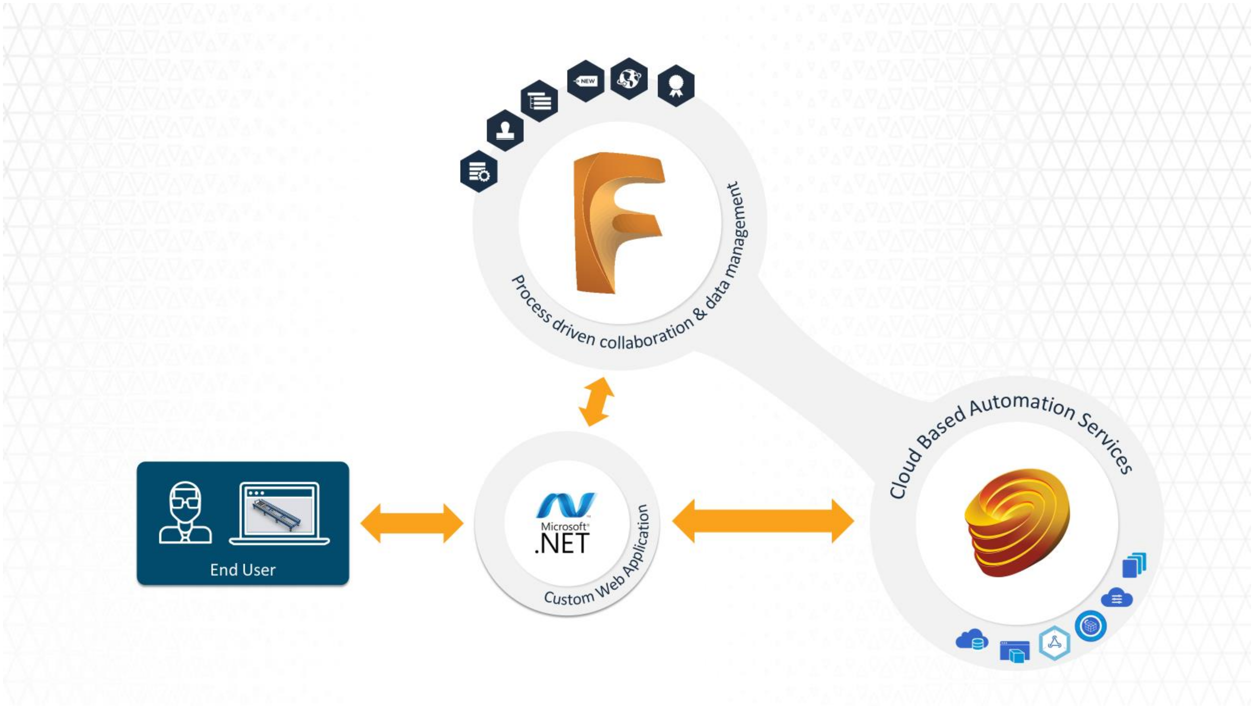
The next sections of this document will further explore the solution architecture and the used building blocks (including some code samples).

Solution architecture

To connect both platforms and benefit from their powerful functionality, a custom Web Application (deployed on Azure) has been created.

On the backend this application provides a set of webservices that allows us to interact with the Forge and Fusion Lifecycle API's in order to process the request coming from the front end. These webservices are build in C#. The front end webpage interacts with these webservices using JavaScript.

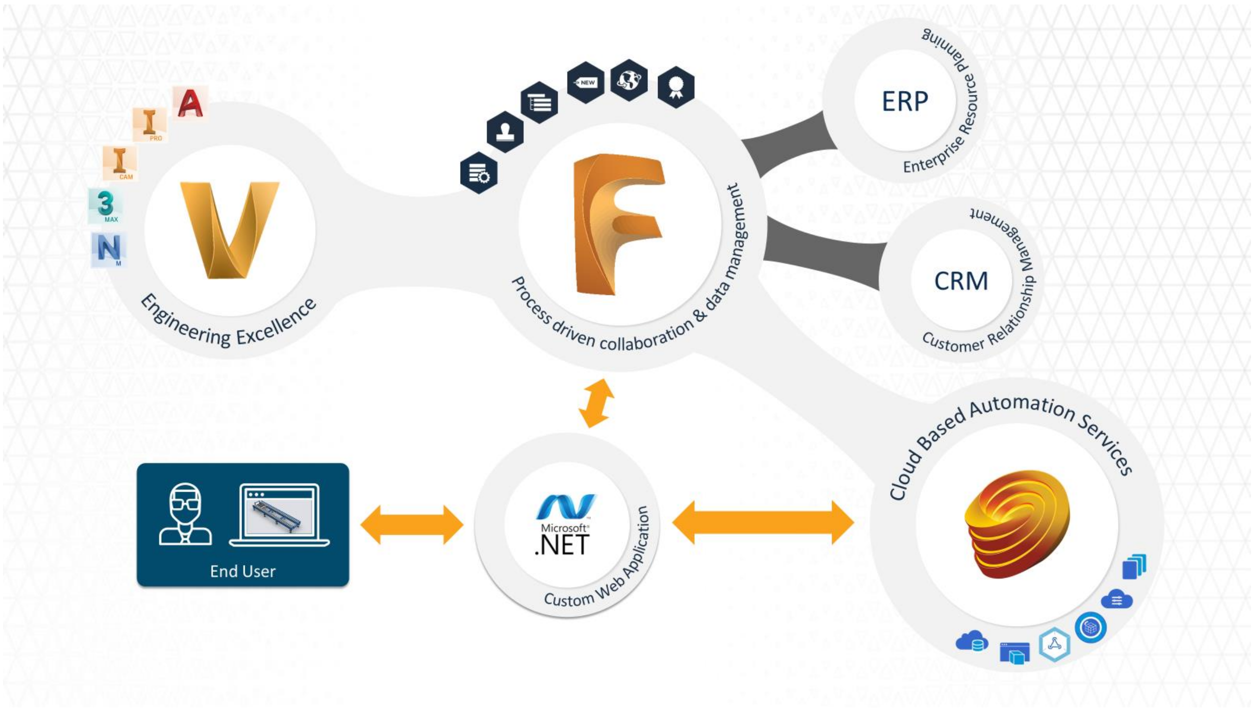
Further on in this hand-out the most important building blocks of the solution are explained more in depth.


Process driven collaboration & data management
Cloud Based Automation Services
End User
Microsoft .NET
Custom Web Application

Scalable Solution: Integrate with other systems

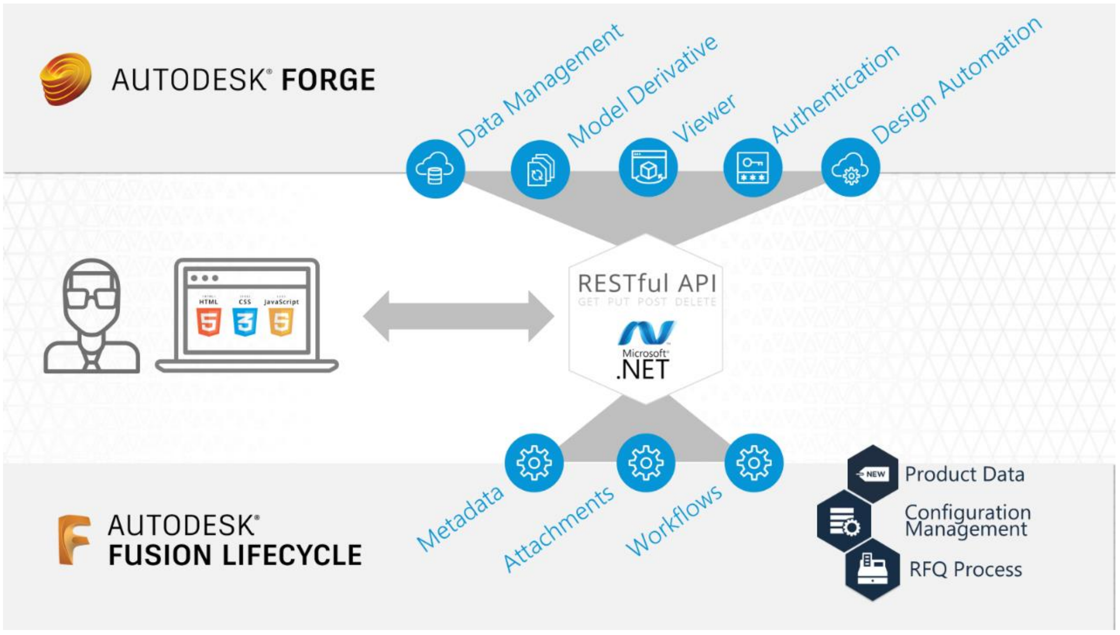
Autodesk Fusion Lifecycle has been build to easily integrate with several business systems. From connecting engineering data from within Autodesk Vault to integrating systems like ERP and CRM.

Autodesk also offers a standard connection to Autodesk Vault, an engineering PDM system. Vault PLM combines Vault Professional with Fusion Lifecycle for enterprise-wide collaboration and product lifecycle management.



Engineering Excellence
Process driven collaboration & data management
ERP
Enterprise Resource Planning
CRM
Customer Relationship Management
Cloud Based Automation Services
End User
Microsoft .NET
Custom Web Application

Building blocks



The system architecture of the solution.

Design Automation for Inventor

The Design Automation API provides the ability to use the core API's of Autodesk Inventor in the cloud, leveraging the scale of the Forge Platform to run automated jobs.

These jobs could be highly repetitive or frequent, or could be larger problems that need large-scale processing power. With the Design Automation API, you can offload that processing to the Forge Platform, which can process those jobs at a much greater scale and efficiency.

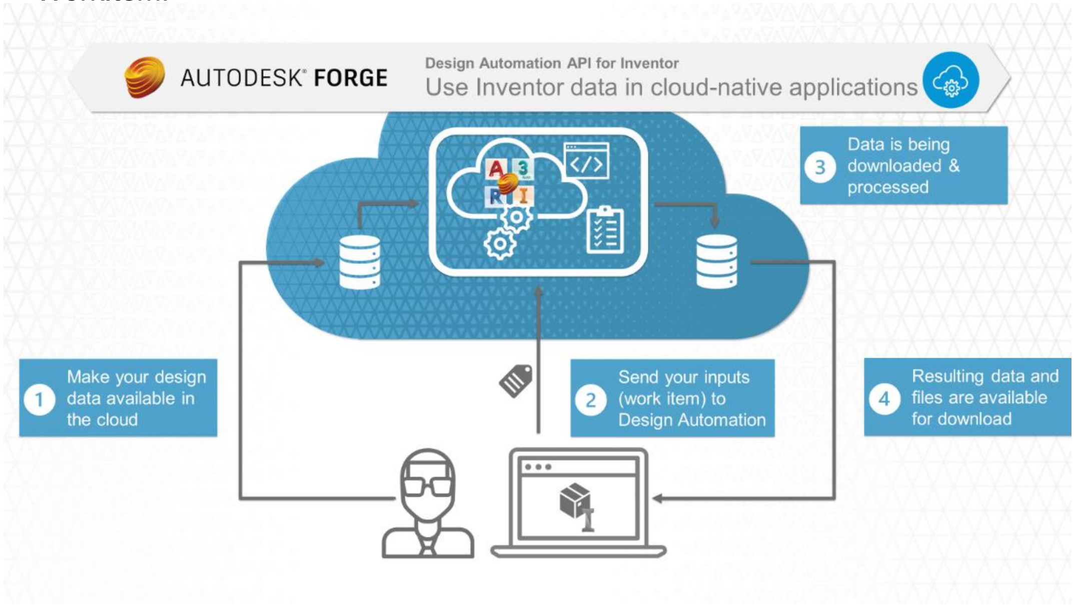
Design Automation API for Inventor provides access to the full Inventor API ("server" only), including functionality such as:

- Modify parameters and features
- Update Drawings
- Use iLogic rules or custom code
- Provide results in any Inventor-supported formats (SAT, DWF, …)

To start coding you can use the Design Automation for Inventor Visual Studio template. This template contains a working Plugin project where just your own business logic needs to be put to. The second project included in the template allows you to locally debug your plugin in case you need to fine tune your plugin or investigate it's behavior.

Notable other resources can be found here:
Working example of the complete solution using Design Automation API for Inventor can be found here.
AU 2018 Class: Bringing your Inventor Add-in into Forge Design Automation
Andrew Akenson's GitHub: https://github.com/akenson
Forge Developer's Guide

To work with the Design Automation API you will need to first Authenticate with Forge, create an AppBundle & Activity, upload your data to a Bucket and finally submit a WorkItem:


AUTODESK® FORGE
Design Automation API for Inventor
Use Inventor data in cloud-native applications
3 Data is being downloaded & processed
1 Make your design data available in the cloud
2 Send your inputs (work item) to Design Automation
4 Resulting data and files are available for download

In the next section we will provide you with some code snippets of the most important building blocks of this workflow.

Disclaimer: THIS SAMPLE CODE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Authenticate with Forge

Forge Authentication (OAuth) is the open standard used across the Forge Platform for token-based authentication and authorization. By making a call to the OAuth REST endpoint with the given credentials, a token will be returned that can be used across your app code. The same token can be used to authenticated with the Fusion Lifecycle platform.

To get proper credentials ("client ID" and "client secret"), you will need to register your app on https://forge.autodesk.com

```
/// <summary>
/// Get the access token from Autodesk
/// </summary>
private static async Task<dynamic> Get2LeggedTokenAsync(Scope[] scopes)
{
    // Connect to the Authentication API
    TwoLeggedApi oauth = new TwoLeggedApi();
    string grantType = "client_credentials";

    // Request bearer (Client ID and Client Secret are stored in the Web.Config)
    dynamic bearer = await oauth.AuthenticateAsync(
      GetAppSetting("FORGE_CLIENT_ID"),
      GetAppSetting("FORGE_CLIENT_SECRET"),
      grantType,
      scopes);
    return bearer;
}

/// <summary>
/// Reads appsettings from web.config
/// </summary>
public static string GetAppSetting(string settingKey)
{
    return Environment.GetEnvironmentVariable(settingKey);
}
```

In the above code sample you will see that we have stored our credentials in the web.config of our app. Our application is using **two-legged authentication** to get a token, so only the client ID and secret are needed. You have also the possibility to implement **three-legged authentication**, where an additional user login is needed.

You will also need to provide the scope for your token. This is actually defining the context in which that application may act. E.g. only data read permissions. A list of scopes can be found here: https://forge.autodesk.com/en/docs/oauth/v2/developers_guide/scopes/

Create Bucket

A bucket in the Forge Data Management platform is used to store objects (files). In our case we are using the bucket to store the input and output data for the Design Automation service.

```
/// <summary>
/// Create Bucket
/// </summary>
private async Task<IActionResult> CreateBucket()
{
    // Authenicate using the Forge Authenthication
    dynamic oauth = await OAuthController.GetInternalAsync();

    // Initiate bucket name
    string bucketkey = "inventorilogicdapat" + nickName.ToLower();

    // Connect to Bucket API
    BucketsApi bucketsApi = new BucketsApi();
    bucketsApi.Configuration.AccessToken = oauth.access_token;

    // Check if Bucket already exists
    dynamic buckets = await bucketsApi.GetBucketsAsync();
    bool bucketExists = buckets.items.ToString().Contains(bucketkey);
    if (!bucketExists)
    {
      // Create new bucket
      PostBucketsPayload postBucket = new PostBucketsPayload(bucketkey, null,
      PostBucketsPayload.PolicyKeyEnum.Transient);
      dynamic newbucket = await bucketsApi.CreateBucketAsync(postBucket);
    }
    return Ok();
}
```

Create Appbundle

AppBundles define the custom programs (add-in) or scripts to be executed on one or more documents on a selected engine.

In a first step you will need to upload your appBundle to the Design Automation platform. In our case the appBundle includes the custom dll we've build to process the Inventor data. The code used for this can be found in the github repository.

During the creation of the AppBundle you also need to specify the given engine you want to use. Within our app we are using Inventor as engine, referred to as : `"Autodesk.Inventor+2021"`

```
// App Bundle Name
string appBundleID = string.Format("{0}.{1}+{2}", nickName, APPNAME, ALIAS);

// Check if App Bundle Exists
Page<string> appBundles = await _designAutomation.GetAppBundlesAsync();
if (!appBundles.Data.Contains(appBundleID))
{
  // Local App Bundle Package
  if (!System.IO.File.Exists(LocalAppPackageZip)) throw new Exception("Appbundle not
found at " + LocalAppPackageZip);

   // Create App Bundle
 AppBundle appBundleSpec = new AppBundle()
 {
    Package = APPNAME,
    Engine = EngineName,
    Id = APPNAME,
    Description = string.Format("Description for {0}", APPNAME),
 };
 AppBundle newApp = await _designAutomation.CreateAppBundleAsync(appBundleSpec);
 if (newApp == null) throw new Exception("Cannot create new app");

 // create alias pointing to v1
 Alias aliasSpec = new Alias() { Id = ALIAS, Version = 1 };
 Alias newAlias = await _designAutomation.CreateAppBundleAliasAsync(APPNAME, aliasSpec);

 // upload the zip with .bundle
 RestClient uploadClient = new RestClient(newApp.UploadParameters.EndpointURL);
 RestRequest request = new RestRequest(string.Empty, Method.POST);
 request.AlwaysMultipartFormData = true;
 foreach (KeyValuePair<string, string> x in newApp.UploadParameters.FormData)
 request.AddParameter(x.Key, x.Value);
 request.AddFile("file", LocalAppPackageZip);
 request.AddHeader("Cache-Control", "no-cache");
 await uploadClient.ExecuteTaskAsync(request);
 }
```

Create Activity

An Activity associate one or more AppBundles to one or more documents, so that a set of custom plug-ins or scripts can be executed against those documents on a given engine.

In our case we are going to use our previously uploaded appBundle that will run on the Inventor Core Console with a given specification:

```
// define the activity
string commandLine = $"$(engine.path)\\InventorCoreConsole.exe /al
\"$(appbundles[{APPNAME}].path)\"";

Activity activitySpec = new Activity()
{
   Id = ACTIVITY_NAME,
   Appbundles = new List<string>() { string.Format("{0}.{1}+{2}", nickName, APPNAME,
ALIAS) },
   CommandLine = new List<string>() { commandLine },
   Engine = EngineName,
   Parameters = new Dictionary<string, Parameter>()
   {
    { "inputJson", new Parameter() { Description = "input json", LocalName =
"params.json", Ondemand = false, Required = false, Verb = Verb.Get, Zip = false } },
    { "ResultCOL", new Parameter() { Description = "output Collaboration file", LocalName
= outputCOLFile, Ondemand = false, Required = true, Verb = Verb.Put, Zip = false } },
    { "ResultPNG", new Parameter() { Description = "output IMAGE file", LocalName =
outputPNGFile, Ondemand = false, Required = true, Verb = Verb.Put, Zip = false } },
    { "ResultSAT", new Parameter() { Description = "output SAT file", LocalName =
outputSATFile, Ondemand = false, Required = true, Verb = Verb.Put, Zip = false } },
    { "ResultDWF", new Parameter() { Description = "output DWF file", LocalName =
outputDWFFile, Ondemand = false, Required = true, Verb = Verb.Put, Zip = false } },
    { "ResultPDF", new Parameter() { Description = "output PDF file", LocalName =
outputPDFile, Ondemand = false, Required = true, Verb = Verb.Put, Zip = false } }
    }
 };
Activity newActivity = await _designAutomation.CreateActivityAsync(activitySpec);

// specify the alias for this Activity
Alias aliasSpec = new Alias() { Id = ALIAS, Version = 1 };
Alias newAlias = await _designAutomation.CreateActivityAliasAsync(ACTIVITY_NAME,
aliasSpec);
```

In the above sample code you will see that we are passing the parameters to Design Automation via a json file (params.json) and getting several result files back from our appBundle: .collaboration, .png, .sat, .dwf, .pdf

Execute Workitems

After the AppBundle is uploaded and the different activities are defined, you can start using this by submitting workitems to the Design Automation service. A workitem can be seen as a job ticket that will be placed on a queue and later picked up by an engine.

The definition of the workitem has to constructed before submitting the Job:

```
WorkItem workItemSpec = new WorkItem()
{
    ActivityId = qualifiedActivityId,
    Arguments = new Dictionary<string, IArgument>()
{
    { "inputJson",  inputJsonArgument },
    { "ResultCOL", outputCOLFileArgument },
    { "ResultPNG", outputPNGFileArgument },
    { "ResultSAT", outputSATFileArgument },
    { "ResultPDF", outputPDFFileArgument },
    { "ResultDWF", outputDWFFileArgument },
    { "onComplete", new XrefTreeArgument { Verb = Verb.Post, Url = callbackUrl } }
}
};
WorkItemStatus workItemStatus = await _designAutomation.CreateWorkItemAsync(workItemSpec);
```

The `inputJsonArgument` is a json string with all model parameters that need to be used for configuring the model in Inventor.

The `outputDWFFileArgument` is an XrefTreeArgument that will contain the location for the specific output file:

```
//  output DWF file
XrefTreeArgument outputDWFFileArgument = new XrefTreeArgument()
{
    Url =
string.Format("https://developer.api.autodesk.com/oss/v2/buckets/{0}/objects/{1}",
bucketkey, outputDWFFile),
    Verb = Verb.Put,
    Headers = new Dictionary<string, string>()
       {
           {"Authorization", "Bearer " + oauth.access_token }
       }
};
```

Fusion Lifecycle
Autodesk Fusion Lifecycle offers a powerful set of REST API's built specifically to execute programmable services via the web.

In the following section you will find some code snippets, using the V3 version of the Fusion Lifecycle Rest API. A brief description and documentation of the webservice are available here:
https://help.autodesk.com/view/PLM/ENU/?guid=FLC_RestAPI_Read_Me_First_html

Rest Calls to Fusion Lifecycle

To execute REST calls to the Fusion Lifecycle API, a generic function has been created. This function will first get a authentication using the same method we have discussed before and then send a request (eventually with a specific requestbody) to Fusion Lifecycle.

```
private async Task<dynamic> ExecuteFLCRequest(string url, Method method, string
AcceptHeader, string body = null)
{

    // Get OAuth token
    dynamic oauth = await OAuthController.GetInternalAsync();
    string Authorisation = "Bearer " + oauth.access_token;

    // Create new rest client
    RestClient FLCClient = new RestClient(BaseURL);
    RestRequest request = new RestRequest(url, method);

    // Set Authenication headers
    request.AddHeader("Accept", AcceptHeader);
    request.AddHeader("Authorization", Authorisation);
    request.AddHeader("x-user-id", FLCUser);
    request.AddHeader("X-Tenant", Tenant);

    // Request Body (if there is one)
    if (!string.IsNullOrEmpty(body))
    {
        request.AddHeader("Content-Type", "application/json");
        request.AddParameter("", body, ParameterType.RequestBody);
        request.RequestFormat = DataFormat.Json;
    }

    // Execute request
    IRestResponse response = await FLCClient.ExecuteTaskAsync(request);

    return response;
}
```

Get Items

This webservice will be used to get all items from a given workspace. This webservice uses the above function to send the request to Fusion Lifecycle:

```
[HttpGet]
[Route("api/FLC/GetWorkspaceItems/{wsid}")]
public async Task<dynamic> GetworkspaceItems(string wsid = null)
{

    string url = $"/api/v3/workspaces/{wsid}/items";
    string AcceptHeader = "application/json";

    IRestResponse response = await ExecuteFLCRequest(url, Method.GET, AcceptHeader);

    return response.Content;

}
```

Other Fusion Lifecycle services

Other code samples (creating Items, using tableaus, uploading files) can be found in the demo application on GitHub: https://github.com/avondtp/InventorDA-FLC

Code Samples References

Our solution is based on one of the code samples of the Forge team: Rim Configurator. All credits go to Sajith Subramanian for sharing this sample on GitHub: https://github.com/sajith-subramanian/forge-rimconfigurator-inventor

Another source of inspiration was the Design Automation Demo app: https://inventor-config-demo.autodesk.io/. Code for this app is also freely available on GitHub: https://github.com/Autodesk-Forge/forge-configurator-inventor

If you need more inspiration for forge development projects, please feel free to visit the forge showroom website: https://forge-showroom.autodesk.io/

We have built our own webservice to interact with both Platforms. In our demo environment the webservices are deployed on an Azure Webserver, but they can be hosted elsewhere. The code is basically build in C# and client-side JavaScript to interact with the webpages.

Note: The source code of demo application is available for download on GitHub: https://github.com/avondtp/InventorDA-FLC

FAQ

In this section we want to address the most commonly asked questions about our class:

1. **How can we actually get this solution to work?**

This is not an out of the box solution, so as a customer you want to engage with an integration partner to get these workflows implemented. As part of this class we will make the sample code available on GitHub: https://github.com/avondtp/InventorDA-FLC

2. **What is required to make this solution work?**

As you may have noticed we are using several technology platforms:

- For the stakeholders that are part of the business processes, a subscription to Fusion Lifecycle is required.
- Some of the Forge building blocks (like the Design Automation) that we are using consuming cloud credits. You will need to make sure you have enough cloud credits in your forge account to run these services
- And finally the web portal and webservices need to be hosted on a cloud platform like Azure or amazon.

3. **Do you have customers using this or a similar solution?**

This solution is based on the customer conversations we have right now. A lot of our customers are currently thinking of implementing such workflows in their existing business processes. Our partners are also looking at building specific offerings that address our customer needs by using our Cloud platform.

4. **As we are building custom solutions based on standard services, what skills are required to develop such a solution?**

There are two sides on this: On one hand you will need the skills to understand the business processes and needs. You need to be able to map this needs to a solution architecture.
On the other hand, if you want to tie all ends together using Autodesk Forge, you will need to have some programming skills. Especially a good knowledge of Web Services and Cloud APIs.

A good starting point could be: https://customersuccess.autodesk.com/articles/get-started-with-forge-using-these-learning-paths