Using Dynamo to Assist with a Solar Study
Fernando Pavon
Autodesk

Learning Objectives

- Using Dynamo and include some simple examples of using Python
- Using RevitAPI functionality to control the sun
- Working with curtain panels that react to the sun’s location
- Working with 2 point adaptive components that follow the sun
- Working with adaptive components that open and close depending on the sun’s location.

Description

This session will be a hands-on introduction to Dynamo as a visual programming tool, it includes a bit of Python and a little Revit API. With this knowledge we will control solar panels that follow the sun, to allow designers to deside on the optimal distance, rotation and layout of the panels. We will also look at how to use parametric curtain panels and adaptive components that open and close depending on the location of the sun, this will help designers to design buildings that will react with it’s environment to optimize energy consumption.

Speaker

Over thirty years experience working as an Architectural Technician managing production of working drawings using CAD systems specialising in automating CAD tasks from the early days of AutoCAD using LISP and later using Revit to automate the BIM process using Dynamo, Python, Microsoft C#.Net and the Revit API.

Tools and Applications

- Revit Version 2021
- Dynamo
- Revit API

This handout goes with a video that will explain in more details what the dynamo scripts are doing.
Lab 1 – Controlling the Sun

In this lab we will begin with a simple task of drawing a line that will follow the sun.

- Open the Revit model SunDirection_2021.rvt

Select Manage in the Revit Ribbon and click the Dynamo command button. This will open Dynamo. You can follow the lab step by step or open the finished Dynamo script which for this lab is called SunDirection.dyn.

- Search for the `SunSettings.Current` node in the Dynamo Node Library.
- Place a `SunSettings.SunDirection` node and connect them as show in the image.

Notice we group nodes and colour code them. This makes it easier to read a complex script.
• The SunSettings.SunDirection node outputs a Vector which represents the direction of the sun. A Vector is similar to a line, it has a start point and a direction, it is often used to move in 3D space by direction.

• Place a Line.ByStartPointDirectionLength node.
• Place a Point.Origin node, this is the same as a Point node with x=0, y=0, z=0 connect it to the startPoint.
• Connect the SunDirection Vector to the direction input port.
• For the length property, use a Number node or a Code Block, or a NumberSlider if you want to vary the length of the line an enter a value like 7000.

You can now set the sun location by dragging the sun or going to the Sun Settings dialog in Revit, but why not create a node that will help us control the sun in revit by dragging a NumberSlider. The Number Slider will represent a frame, or sun location in 30 minute intervals.

Place a Number Slider and a Python Script node. Double click the python node and type the following Python code. If you don’t want to type you can cut and paste from the file python.py saved in the Lab folder.

```python
# Every line starting with a # sign is ignored by Python
# usually used to place comments
# no need to type the comments for your code to work but it’s
# always advisable to comment your code
# The following lines in green are used to load libraries that
# we will need in our code,
# like the reference to the RevitAPI

# Load the Python Standard and DesignScript Libraries
import sys
import clr

# Import DocumentManager and TransactionManager
#Transactions are required to make changes to the Revit model
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager
```
# Import RevitAPI
clr.AddReference("RevitAPI")
import Autodesk
from Autodesk.Revit.DB import *

# The following is our first line of code which gets active revit
# document into a variable called doc
doc = DocumentManager.Instance.CurrentDBDocument

# inputs from Dynamo to the Python node are saved in a list called IN[ ]
# IN[0] is the first input value IN[1] would be the next…etc
# a day from sunrise to sunset is divided into frames or time intervals
frame = IN[0]

# The following line gets the sunSettings from the Revit model’s current view
sunSettings = doc.ActiveView.SunAndShadowSettings

# we are going to change the model so we begin a transaction in the doc (document)
TransactionManager.Instance.EnsureInTransaction(doc)

# we set for a one day study, time intervals at 30 minutes, and set the active frame
# active frame is the input coming from the Number Slider
sunSettings.SunAndShadowType = SunAndShadowType.OneDayStudy
sunSettingsTimeInterval = SunStudyTimeInterval.Minutes30
sunSettings.ActiveFrame = frame

# all changes done so close the transaction
TransactionManager.Instance.TransactionTaskDone()

# no output required
# when you drag the Number Slider the sun location will move back and forth in
# intervals of 30 minutes.
OUT = 0

Lab 1 is now complete make sure it’s all working. As you slide the Number Slider the sun
should move back and forth from sunrise and sunset. Make the Number slider max and min
values from 1 to the number of 30 minute intervals to sunset. Depending on your project
location and the season it could vary for example a day of 12 hours will require 24 frames.
The Revit API

Behind most Dynamo nodes, especially the ones under the Revit library you can be sure the Revit API is being used. In the previous Python we used the Revit API class SunAndShadowSettings to set the frame interval and to set the current frame, or sun location. Knowing how to use the Revit API will give you special powers to automate Revit to achieve maximum quality, reliability and production efficiency. We will take another peak at the Revit API again in Lab 6.

In the meantime copy the following files in Lab 6:

- RevitAPI_AddIn.addin
- RevitAPI_AddIn.dll

To the following folder:

- C:\ProgramData\Autodesk\Revit\Addins\2021

The next time you run Revit you should have an additional Tab called Solar Study with the following command buttons shown in the image:

The Revit API is fully documented in the website [www.revitapidocs.doc](http://www.revitapidocs.doc)
Lab 2 Solar Panels – Follow the Sun

To save time you can use the adaptive component called SolarPanel.rfa. But just for your interest, the following are the steps taken to create a 2 x point adaptive component:

- Create a Generic Family
- Place two reference points
- Select the points and make them adaptive points
- Draw a reference line between the two points
- Place a point on the reference line
- Select the point – the plane is selected
- Draw a circle
- Select the circle and the reference line and click Create Solid Form
- Set the workplane to the top of the reference line
- Draw a solar panel by extruding a 1000 x 1000 square
- Test to make sure it works

- Open the Revit project SolarPanels.rvt
- Open the Dynamo script SolarPanels.dyn

Things to try:
In the Panel location points group, disconnect the input from the x, y and z ports of the Point.ByCoordinates node. Now you should have only one panel at 0,0,0. This is because the default values for z, y and z are 0.

In the Code Block node enter a list of values with the following syntax:

\([-3000, -1500, 0, 1500, 3000]\);

A list of values are placed between square brackets [ , , , ] separated by commas and you must add the semicolon at the end.

Connect the values to the x port. You should now have a row of panels.

Connect the same input list to the y port.

Right click the Auto in the Point.ByCoordinate node and select: Lacing – Cross Product

Use the slider to move the sun back and forth, it should work ok for a single panel or even a row but for 25 panels it may be too slow, then if use the AddIn buttons installed earlier in the Solar Panel Tab of the Revit Ribbon.
Lab 3 Curtain Panels Solar Study

For this lab we create a parametric curtain panel that will open and close depending on the direction angle of the sun.

In the lab folder there is the complete version of the Frame_Panel.rfa. The following are the steps required to create it.

- Create a Revit family using the Curtain panels Family Template.
- Create the Reference Planes as per the illustration
- Create a simple extrusion with a void in the middle
- Ensure to lock the geometry to the Reference Planes
- Set the formulas as shown
- Test and flex, make sure everything is working
• Load the project CurtainPanels_SolarStudy.rvt

• The Curtain Wall has the Curtain Panel set to Frame_Panel
• You can vary the spacing depending on the design.
• Load the Dynamo script **CurtainPanels_SolarStudy.dyn**

Flipping the connectors for max and min will have the opposite effect. Depending on location, season and design intent you may want to block the sun or harness its energy.

• Load the Dynamo script **CurtainPanels_SolarStudy_WithLines.dyn**
• The lines show the vectors used to calculate the angle between the sun direction and the direction the panel is facing.

You can see the perpendicular direction for each panel, the lines for the sun direction are parallel. The angle between the two will give us the ratio between 0.0 and 1.0 which will determine how wide to open the panel.
Lab 4 Adaptive Components

This lab is based on the Al Bahar building in Abu Dahbi which provides an example of smart buildings aiming to control energy from the sun. As you may know Abu Dahbi is a hot place, to say the least, air conditioners are going on all the time, so controlling exposure to the sun is key.

The towers are protected by a shield of smart panels that can be opened or closed depending on the sun’s direction.

This lab begins with the creation of an adaptive component or a curtain panel to be used in a pattern based surface of a mass. The component is based on a 3 x point made into adaptive points. There won’t be enough time to create this component in this lab but it was worth providing it as an example so we have included the Revit family AlBahar_Panel_2021.rfa this component is driven by the parameter Opening as shown:

![Diagram of adaptive component]

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base (report)</td>
<td>3000.0</td>
</tr>
<tr>
<td>Center (default)</td>
<td>0.3333333</td>
</tr>
<tr>
<td>Height (default)</td>
<td>2598.1</td>
</tr>
<tr>
<td>Hr (default)</td>
<td>1732.1</td>
</tr>
<tr>
<td>Hr_NCP (default)</td>
<td>0.807468</td>
</tr>
<tr>
<td>Hr_base (default)</td>
<td>1502.5</td>
</tr>
<tr>
<td>Lh (default)</td>
<td>85.6</td>
</tr>
<tr>
<td>Opening (default)</td>
<td>0.9</td>
</tr>
<tr>
<td>Ps (default)</td>
<td>0.3172</td>
</tr>
<tr>
<td>Ps_NCP (default)</td>
<td>0.331662</td>
</tr>
<tr>
<td>Panel Thickness (default)</td>
<td>10.0</td>
</tr>
<tr>
<td>Ratio (default)</td>
<td>0.000000</td>
</tr>
<tr>
<td>h (default)</td>
<td>186.0</td>
</tr>
</tbody>
</table>

Opening = h / Height

Opening = sqrt((Hr * 2) - (Ps * 2))

Opening = h ^ (1 - Ratio)

Opening = Opening / 1 mm

Opening = Height / 3
Open the Revit sample model AdaptiveModel.rvt

Open the Dynamo file AdaptiveModel.dyn

Use the Solar Panel command buttons to move the sun back and forth to see how the parametric components open and close. The Python node takes all the angles and converts them into the opening ratio between 0.0 and 1.0. See video for more explanation about the Dynamo script.
Lab 5 Combined Model

This lab simply combines all the scripts into a single model. This model is used for training purposes only it does not aim to be an example of aesthetic design.

Open SampleModel_2021.rvt

Open CombinedDynamo.dyn

Depending on the power of your machine, this could be slow, everytime you advance to a different frame the Dynamo script has to re calculate all the components to adjust for the new sun position. You can of course capture each frame and create an animation.

AU2020Project.mp4
Lab 6 RevitAPI AddIn

In this lab we provide a simple example with sample code that shows how to setup a Revit AddIn. These files were prepared in Microsoft’s Visual Studio and are written using .NET C#.

Normally you would create a new Class Library (.NET Framework) project in Visual Studio but for this lab we will open the sample project created for this lab.

Use Visual Studio to open the **RevitAPI_AddIn.sln** solution file. One solution can contain many projects, in this case we have just one.

- Once open take a look at the Solution Explorer
- Under References you will see the references made to the **RevitAPI** and **RevitAPIUI** that are required to access the Revit API functionality within your code.
- Under Resources we have added the three 32x32 pixel images required for the command buttons.
- You will also find the **App.cs** class. This is the main application class which is the entry point for this project.
- When Revit launches it will look under the Revit addins folder under the correct version, in this case 2021, for all the addin files.
- The addin file is an XML text file, it tells Revit where to find the application which in this case is called **RevitAPI_AddIn.dll** this file contains within it all the classes in the project.
RevitAPI_AddIn.addin

Below is an example used for this project.

This is the first file Revit opens, it sees that the assembly is called RevitAPI_AddIn.dll

It also sees that the startup class is App in the assembly.

```xml
<xml version="1.0" encoding="utf-8"?>
<AddIn>
  <AddIn Type="Application">
    <Name>RevitAPI Solar Study AU 2020</Name>
    <Assembly>RevitAPI.AddIn.dll</Assembly>
    <FullClassName>RevitAPI.AddIn.App</FullClassName>
    <ClientID>28593628-CC89-4767-98CA-166F4A8D759</ClientID>
    <VendorId>Autodesk</VendorId>
    <VendorDescription>Consulting</VendorDescription>
  </AddIn>
</RevitAddIns>
```

**App.cs**

The application class is this project is called App.cs

It inherits from IExternalApplication for the RevitAPI

The first thing that Revit does is run the OnStartup function

Within that function Revit creates a Ribbon Tab called Solar Study

Then creates a Panel called Solar Study

Then creates three command buttons, (just one shown in this sample)

```csharp
using System;
using System.Reflection;
using Autodesk.Revit.UI;

namespace RevitAPI.AddIn
{
    public class App : IExternalApplication
    {
        public Result OnStartup(UIControlledApplication application)
        {
            string assemblyPath = Assembly.GetExecutingAssembly().Location;
            string startupTab = "Solar Study";

            // Create Tab
            application.CreateRibbonTab(startupTab);

            // Solar Study Panel
            RibbonPanel panel1 = application.CreateRibbonPanel(startupTab, "Solar Study");

            PushButton button = panel1.AddItem(new PushButtonData("CP Solar Study", "Sun Reset", assemblyPath, "RevitAPI.AddIn.SunReset Command") as PushButton;
            Uri urlImage = new Uri($"pack://application:,,,/RevitAPI.AddIn;component/Resources/Brightness.png", UriKind.Absolute);
            BitmapImage image = new BitmapImage(urlImage);
            button.LargeImage = image;
            button.ToolTip = "Solar Study";
        }
    }
}
```

Every command button has its code in a separate class:

This project has three commands:

- SunBack_Command.cs
- SunForward_Command.cs
- SunReset_Command.cs
Each command class inherits from the RevitAPI class IExternalCommand. Each command class must have the Execute function which runs the code for that command button.

```csharp
using System;
using Autodesk.Revit.Attributes;
using Autodesk.Revit.DB;
using Autodesk.Revit.UI;
	namespace RevitAPI_AddIn
{
    [Transaction(TransactionMode.Manual)]
    public class SunResetCommand : IExternalCommand
    {
        public Result Execute(ExternalCommandData commandData, ref string message, ElementSet elements)
        {
            return Result.Succeeded;
        }
    }
}
```

### Sun Reset Command

The Sun Reset command gets the sun settings from the active document and active view. It gets the sunrise and sunset times for a specific day. Remember any changes to the model must be made within a Transaction. We set the settings to a One Day Study, we set the start and end times one hour after sunrise and one hour before sunset. It sets the active frame to 1.

```csharp
//get current settings from view
SunAndShadowSettings sunSettings = m_doc.ActiveView.SunAndShadowSettings;

//set sunrise on April 28, 2011
DateTime sunrise = sunSettings.GetSunrise(DateTime.SpecifyKind(new DateTime(2011, 4, 20), DateTimeKind.Local));

//set sunset on April 22, 2011
DateTime sunset = sunSettings.GetSunset(DateTime.SpecifyKind(new DateTime(2011, 4, 22), DateTimeKind.Local));

Transaction tx = new Transaction(m_doc);

sunSettings.SunAndShadowType = SunAndShadowType.OneDayStudy;

//frames start 1 hour after sunrise on April 28, 2011
sunSettings.StartDateTime = sunrise.AddHours(1);

//frames end 1 hours before sunset on April 22, 2011
sunSettings.EndDateTime = sunset.AddHours(-1);

if (sunSettings.IsTimeIntervalValid(SunStudyTimeInterval.Minutes30))
    sunSettings.TimeInterval = SunStudyTimeInterval.Minutes30;

//check for validity of start and end times
if (!sunSettings.IsAfterStartAndTime(sunSettings.EndDateTime) 
    && sunSettings.IsBeforeEndAndTime(sunSettings.StartDateTime))
    TaskDialog.Show("Error", "Start and End dates are invalid");

// turn on display of the sun path
m_doc.ActiveView.get_Parameter(BuiltInParameterVIEW_GRAPH_SUN_PATH).Set(1);

sunSettings.ActiveFrame = 1;

tx.Commit();
```
Move Sun Back and Forth

These commands are simple. We get the sun settings for the active document and active view and add or subtract the active frame by 1.

```java
SunAndShadowSettings sunSettings = m_doc.ActiveView.SunAndShadowSettings;

Transaction tx = new Transaction(m_doc);
tx.Start("Move Sun Forward");

sunSettings.ActiveFrame += 1;

tx.Commit();

SunAndShadowSettings sunSettings = m_doc.ActiveView.SunAndShadowSettings;

Transaction tx = new Transaction(m_doc);
tx.Start("Move Sun Back");

sunSettings.ActiveFrame -= 1;

tx.Commit();
```