# PythonParts for Developers

Created 11/2015 Bert Van Overmeir – SCIA Herk-de-Stad

# Contents

Installation	2
Allplan with Python Integration	2
Installing the development kit for Python	2
Visual Studio 2013	2
PythonTools for Visual Studio	4
Installing the Python Interpreter	5
Basic editing of PythonParts in Allplan	6
Opening your first PythonPart	6
Editing PythonPart parameters	7
Opening a .pyp file	7
Editing a .pyp file	8
Externally editing parameters for Allplan10	0
Advanced Editing of PythonParts in Visual Studio1	1
Setting up the workspace1	1
Allplan libraries1	1
Python Project in Visual Studio1	1
Analyzing an example script1	3
Generating PythonDoc1	3
Before we start14	4
Basic PythonPart structure	4
Defining inherited methods1	5
Referring to parameters from the python script1	6
Import statements and python philosophy1	6
Simple column script example1	7

# Installation

# Allplan with Python Integration

Install Allplan with Python support through the installer package provided. After a normal installation has been executed, a few extra additions in relation to PythonParts have been added to Allplan. The respective paths can be found below. We will discuss each one of them more in detail later on the tutorial series.

Firstly, examples of PythonParts can be found under:

# X:\ProgramData\Nemetschek\(...)\ETC\Examples\PythonParts

Please refer to "testing" for more information about the examples. The **Allplan Python libraries** that we will be using are located under:

X:\Program Files\Allplan\(...)\Prg

It is important that you remember this pat as we are going to need it further on installation. Thirdly, **scripts written in Python** can be found under:

*X*:\*ProgramData*\*Nemetschek*\(...)\2016\*ETC*\*PythonPartsScripts* 

It is important that you do not mover any of the files, nor rename the scripts. In the example files, Allplan refers to certain scripts. Execution will fail upon changing names.

# Installing the development kit for Python

To use any of the scripts within Allplan, we do not need to install development kit. If your aim is to use existing PythonParts, you can skip this part and move to "use of PythonParts".

#### Visual Studio 2013

PythonParts are best developed in Visual Studio. Although development is, of course, not limited to this tool, we strongly recommend to use visual studio as this tutorial series and the plugins provided will use Visual Studio too.

To install Visual Studio 2013, visit the link below.

https://www.visualstudio.com/downloads/download - visual - studio - vs

#### Make sure you pick Visual Studio 2013 Community Edition with Update 5 as so:

Visual Studio 2013	Note: If you already have Visual Studio Community 2013 (original release version) and run this download, only Update 5 is installed. If you don't have Visual Studio Community 2013 and run this download, both Visual Studio
<ul> <li>Community 2013</li> </ul>	Community 2013 and Update 5 are installed. In either case, Visual Studio 2013 Language Packs (original release
Ultimate 2013	versions) can also be installed.
Premium 2013	Release notes System requirements
Professional 2013	SHA-1 values
Test Professional 2013	Choose format:      Web installer      ISO
Express 2013 for Desktop	How to install files offline
Express 2013 for Web	Download
Express 2013 for Windows	Visual Studio Community 2013 with Update 5 - English
Visual Studio 2013 Update 5	visual studio community zors with opdate 5 - English

Figure 1: downloading visual studio

Picking another edition of Visual Studio is not wrong, but it is however problematic for installing the Python Tools later on. Download the web installer as we are only going to need a small portion of the full capabilities of Visual Studio. Upon installation at the Optional Features dialogue, it is safe to **untick all the optional features except for the web developer tools**, as they might come in handy for report development in Allplan.



Figure 2: selecting the right installation options in Visual Studio

Now click install. Be aware that installation can take **up to two hours**. So be prepared to do some other work in the meantime. After the installation of Visual Studio has completed, verify if the program starts. You should get the following screen.

	Start Page - Microsoft Visual Studio				P = 0 :
	EDIT VIEW DEBUG TEAM TOOLS TE				Sign in
	itart Page 🗢 🗙			<ul> <li>Solution Explorer</li> </ul>	- B 3
Mar				- 00G ¥	
Server Explorer To	Community 2013	Discover Visual Studio Community 201	3		
blbox		Get the most from Visual Studio with these resources:	What's new on Microsoft Platforms		
	Start	New to Visual Studio? Check out coding tutorials and sample projects Get training on new frameworks, languages, and technologies	Windows Microsoft Azure		
	New Project Open Project Open from Source Control	Create a private code repo and backlog for your project See how easy it is to get started with cloud services Discover ways to extend and customize the IDE	✓ ASP.NET and Web Windows Phone ↓		
		Connect to Azure			
	Recent	Learn more about Azure			
	PythonParts	Connect 🕣			
		Relocate the What's New information			
		Announcements			
		Microsoft Connect(); // 2015 Developer Event Set for November	18-19		
		Thursday, November 5, 2015 Microsoft is a developer company, so there's nothing we love more than connecting future. I benefore invite you to set your calendar for November 18-19 for Connect (): J live and for free, from New York City to developers around the world.	with developers to share our latest tools, technologies and plans for the // 2015 – when Microsoft hosts its premier fall developer event, streamed		
		Cross-Database Queries in Azure SQL Database			
		Thursday, November 5, 2015			
		We are happy to announce a number of significant improvements to elastic database supports querying across databases in Azure SQL Database. This makes possible come into a local table.			
		C++ Memory Profiling Tools in Visual Studio 2015			
		Thursday, November 5, 2015 As announced in an aerlier blog post, Visual Studio 2015 hosts a new set of memory p applications. The new debug-time profiler runs during your debugging session and a view the heap contents during the current debugger break state. This provides for a n	flows you to take snapshots anytime, such as at a breakpoint, and also		
		New Java documentation subsite for Visual Studio Online			
		Thurrday Minuarshar 5, 2015		<b>~</b>	
	Dutput	· [ S ] 5 5 1 2 1 2 1 10	*	7 ×	
	Show output from:	•   F   F F F   E   D			
				Code Analysis Solution Explorer Te	earn Explorer Class View
Ready					

Figure 3: Visual Studio Interface

# PythonTools for Visual Studio

The Required Python tools for Visual Studio should have been provided by your local Allplan dealer. If not, please contact them for more information. Alternatively you can search for the python tools via the link below, and search for **version PTVS 2013 BETA (.30811.01)**.

https://www.visualstudio.com/en - us/features/python - vs.aspx

It is of key importance that you download the correct version of the Python Tools. The wrong version will not work with your Visual Studio installation.

Before installing, you should disable the verification of files, this can be done by editing the registry. The file should also be provided within the installation package you got from the Allplan dealer. If this is not the case, do not continue installation before acquiring them.

Registry Editor	
Adding information can unintentionally change or delete values and cause components to stop working correctly. If you do not trust the source of this information in C:\Users\bertvo\Documents\python\PTVS.30811.01\30811.01\EnableSkipVerification.reg, do	
not add it to the registry. Are you sure you want to continue?	Size
Yes No	14 KB 7 KB
EnableSkipVerification     18/07/2015 17:36     Registration Entries	14 KB

Figure 4: skipping the verification process of the installation, click yes to continue

After adding the entries to the registry, you are free to install the Python Tools for Visual Studio.

闄 Python Tools Beta for Visual	Studio 2013 Setup	_		×
Python Tools for Visual Studio	Welcome to the Python Visual Studio 2013 Setu The Setup Wizard allows you to cha Tools Beta for Visual Studio 2013 fe your computer or to remove it from Next to continue or Cancel to exit t	p Wizar	d ay Python installed o outer. Click	
	Back Ne	ext	Canc	el

Figure 5: installing AFTER the skip verification has been executed

# Installing the Python Interpreter

Upon starting Visual Studio, the Python Tools will be recognized, but you find yourself unable to successfully executing a program as no interpreter has been installed.

To install an interpreter visit:

# https://www.python.org/

Go to downloads, click on windows, scroll down and look for the **python-3.4.3. X64** setup.

After installation, when you start Visual Studio, go to Tools, Options and check if the Python environment is installed.

Options					?	×
Search Options (Ctrl+E)	P	D <u>e</u> fault Environment:	Pytho	n 64-bit 3.4		~
<ul><li>▷ Performance Tools</li><li>▷ Database Tools</li></ul>	^	Show se <u>t</u> tings for:	Pytho	n 64-bit 3.4 🗸	A <u>d</u> d Environment	t
<ul> <li>F# Tools</li> <li>Graphics Diagnostics</li> </ul>		- Environment Settings -				
<ul> <li>▷ HTML Designer</li> <li>▷ NuGet Package Manager</li> </ul>		<u>P</u> ath:		C:\Python34\python.exe		
<ul> <li>Python Tools</li> <li>General</li> </ul>		<u>W</u> indows Path:		C:\Python34\pythonw.exe		
Debug Interactive Window		<u>L</u> ibrary Path:		C:\Python34\lib		
Debugging Environment Options		<u>A</u> rchitecture:		х64		
Interactive Windows SQL Server Tools		Language Version:		3.4		
Text Templating		Path Environment <u>V</u> ar	iable:	PYTHONPATH		
<ul> <li>Web Performance Test Tools</li> <li>Windows Forms Designer</li> </ul>			R	emove Environment		
Workflow Designer	*					
				ОК	Cance	I

*Figure 6: checking if the python environment is installed.* 

If this is not the case, click on add environment and add it manually providing the paths to the python.exe, python.exe and library (view screenshot). If you did a default install, the paths should resemble the paths in the screenshot.

That's it! You are all set up to start working with Python for Allplan!

# Basic editing of PythonParts in Allplan

# Opening your first PythonPart

Every PythonPart in Allplan consists of a few files, but in order to open an existing file, only one file is needed. You might have noticed that Allplan Python examples contain the following files in different folders:

.py files: the python script, we will get into this later .png files: placeholder images, obsolete .pyp files: the most important files that can be used within Allplan

PythonParts can be dragged straight into Allplan. By dragging the *.pyp* file in Allplan, a similar interface to SmartParts is shown.

Home Share	e View			~ 😗	<u>F</u> ile <u>E</u> dit <u>V</u> iew  nsert Forma	at Jools <u>C</u> reate C	ange <u>R</u> epeat <u>W</u> i	ndow <u>?</u>		
→ ~ ↑ 📙 « 20	016_Python > 2016 > ETC > Examples >	PythonParts	Search Python	Parts ,P	🖉 🖼 🗣 🔁 🖥 🌵 🖾				3 🔗 🖉 🖉 0.25 —	
Quick access	Name	Date modified	Туре	Size	PythonPart		4 ×		Central perspective:1	8 J
Desktop #	GeometryExamples	06/11/2015 12:43	File folder		Tools PythonPart Wizards	Library Objects C	onnect Layers			
Downloads #	ReinforcementExamples	06/11/2015 12:43	File folder			Länge	10.0000			
	Folder PythonParts.deu	04/08/2015 09:16	DEU File	1 KB		Länge links	6.0000			
🗄 Documents 🖈	FolderContent_PythonParts.deu	05/08/2015 23:06	DEU File	1 KB		Höhe links	1.0000			
E Pictures 🖈	Beam1	04/08/2015 09:16	PNG File	7 KB		Höhe mitte	1.5000			
OneDrive	Column1	04/08/2015 09:16	PNG File	7 KB						
This PC	ExampleControls	05/08/2015 23:06	PNG File	7 KB		Höhe rechts	1.0000			
This PC	ExampleControlsExpander	12/10/2015 23:07	PNG File	7 KB		Vollquerschnitt links	1.5000			
Network	param01	13/10/2015 16:00	PNG File	1 KB	1	/ollquerschnitt rechts	1.5000			
C60-FIERY	param02	13/10/2015 16:00	PNG File	1 KB	0	bergang I-Profil links	0.3000			
DC2STG01	param03	13/10/2015 16:00	PNG File	1 KB	Üb	ergang I-Profil rechts	0.3000			
DC2STG02	SleeveFoundation_geom1 SleeveFoundation1	13/10/2015 16:00	PNG File	9 KB 7 KB		Flanschbreite unten	0.4000			
DESKTOP-GHD0	SteeveFoundation1	04/08/2015 09:16 04/08/2015 09:16	PNG File PNG File	7 KB 7 KB						
E7450-CHRIS	Stairs1	04/08/2015 09:16	PNG File	7 KB		Flanschhöhe unten	0.3000			
E7450-KATHLEEI	Stairs2	04/08/2015 09:16	PNG File	7 KB	F	lanschschräge unten	0.2000			-tr
E7450-TECLA	Stairs4	04/08/2015 09:16	PNG File	7 KB		Stegbreite	0.2000			J.
FILEBOXV2	Stairs4a	04/08/2015 09:16	PNG File	7 KB		Flanschschräge oben	0.1000			
_	StairsBuilder	04/08/2015 09:16	PNG File	7 KB		Flanschbreite oben	0.6000			
HDSACCBOB1	StartPythonDebug	07/10/2015 17:58	PNG File	7 KB		Flanschhöhe oben	0.2000	5.00	The second	
HDSBCK02	Table1	04/08/2015 09:16	PNG File	7 KB				<b>FCSTUART</b>	ALL AND	
HDSCITRIX1	Table2	10/09/2015 23:06	PNG File	7 KB		Flanschversatz links	0.5000	MIHUW		
HDSICTADC1	WallCorner1	04/08/2015 09:16	PNG File	7 KB		Flanschversatz rechts	0.5000			
HDSICTEST1	WallCorner2	04/08/2015 09:16	PNG File	7 KB						
HDSICTFIL1	Beam1	08/08/2015 01:43	PYP File	4 KB	N			1.1		
HDSICTPRT1	Column1	04/08/2015 09:16	PYP File	2 KB	G					
HDSMRS	ExampleControls	08/10/2015 23:06	PYP File	3 KB						
HDSSTDPC13	ExampleControlsExpander	13/10/2015 16:00	PYP File	2 KB						
HDS-VCENTER1	SleeveFoundation1	13/10/2015 16:00	PYP File	4 KB						
IT-ASSISTANT-B	Stairs1	29/08/2015 01:15	PYP File	20 KB						
KM82D936	Stairs1a Stairs2	29/08/2015 01:15	PYP File PYP File	20 KB						
_	Stairs2	29/08/2015 01:15 29/08/2015 01:15	PYP File PYP File	15 KB 17 KB						
M3800-PATRIZIA	Stairs4	29/08/2015 01:15	PYP File	16 KB						
M4600-JEF	StairsBuilder	29/08/2015 01:15	PYP File	18 KB						
M4800-MARWA	StartPythonDebug	07/10/2015 17:58	PYP File	1 KB						
M6600-FRANCIS	Table1	04/08/2015 09:16	PYP File	2 KB						
M6600-GEERT	Table2	10/09/2015 23:06	PYP File	2 KB						
M6600-GINO	WallCorner1	29/08/2015 01:15	PYP File	5 KB						
M6600-PETERVT	WallCorner2	29/08/2015 01:15	PYP File	7 KB						
M6800-BERT	ExampleControls_deu	07/10/2015 23:07	XML Document	2 KB						
MBPPASCAL	ExampleControls_eng	07/10/2015 23:07	XML Document	2 KB						
PASCALDS										
SEPSERVERBE										
SIXHEADS					Ċ ŵ		Close			
SUPPORT2							•		A 1120	
SUPPORT3					Set the to point $\Delta x$ 0.000	Δy 0.000	Δz 0.000	_ 🗛 🗠 🗛 📃	🗡 🎬 0.010 💌 🖼	•

Figure 7: dragging the PythonPart "Beam" into Allplan which shows a few extra options, just like SmartParts.

In current versions (Allplan 2016.1.0) it is only possible to drag the PythonPart in Allplan and edit it **once**, after you close the tab in Allplan, the elements do get generated and become genuine Allplan objects. This way, a fully functional 3D model of reinforcement can be easily made, but it is not possible to easily change special options like in SmartParts, you would just have to use the standard editing tools like "stretch".

The *.pyp* files do not need to be in the ETC folder in order to be able to work properly. They can be placed anywhere on the computer. It is however important **that the** *.py* **files reside in the correct folder (X:\ProgramData\Nemetschek\\(...)\2016\ETC\PythonPartsScripts)** as the *.pyp* files reference to them. This should already be the case if you are using the examples provided.

# Editing PythonPart parameters

# Opening a .pyp file

Every PythonPart consists of at least two files (more is possible as you write more advanced scripts).

< name of PythonPart > .pyp located wherever you want < name of Script > .py located in PythonPartsScripts

When we open a .pyp file in a text editor, we see something that resembles the following:

```
<?XML VERSION="1.0" ENCODING="UTF-8"?>
<ELEMENT>
<SCRIPT>
<NAME>COLUMN1.PY</NAME>
<GEOMETRYEXPAND>O</GEOMETRYEXPAND>
</SCRIPT>
<PAGE>
<NAME>GEOMETRIE</NAME>
<PARAMETER>
<NAME>COLUMNWIDTH</NAME>
<TEXT>BREITE</TEXT>
<VALUE>600.</VALUE>
<VALUETYPE>LENGTH</VALUETYPE>
</PARAMETER>
```

Every *.pyp* file consists of a fairly straightforward structure written in XML and can be easily edited. The *.pyp* file contains all the parameters that will be later on used to generate the menu within Allplan whilst opening a new PythonPart.

🗟 Column1 - Notepad – 🗆 🗙	Allplan 2016_Python 2016 - New project - <no fileset="">:DF1 _ C X</no>
File Edit Format View Help	Bie Edit View Insert Format Jools Sreate Change Repeat Window 2
xml version="1.0" encoding="utf-8"?	Ø ⊟ G Ø ⊟ ‡ Ø Ø \$ + A + M = = = = = = = = € Ø Ø Ø Q Ø Ø Ø Ø 025 • 1
<element></element>	
<script></td><td>Central perspective. 1</td></tr><tr><td><Name>Column1.py</Name></td><td>Tools PythonPart Wizards Library Objects Connect Layers</td></tr><tr><td><GeometryExpand>0</GeometryExpand></td><td></td></tr><tr><td></script>	
<page></page>	Dicke 0.4000
<name>Geometrie</name>	Höhe 5.000
<parameter></parameter>	A/P
<name>ColumnWidth</name>	Konsolbreite 0.4000
<text>Breite</text>	Konsolhöhe 0.4000
<value>600.</value>	1
<valuetype>Length</valuetype>	Oberkante Konsole 4.0000 🔠
<parameter></parameter>	
<name>ColumnThickness</name>	20
<text>Dicke</text>	
<value>400.</value>	
<valuetype>Length</valuetype>	
<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>	
<name>ColumnHeight</name>	
<text>Höhe</text>	
<value>5000.</value>	
<valuetype>Length</valuetype>	
<parameter></parameter>	
<name>CorbelWidth</name>	
<text>Konsolbreite</text>	
<value>400.</value>	
<valuetype>Length</valuetype>	
	The second se
<name>CorbelHeight</name>	Σ
<text>Konsolhöhe</text>	
<value>400.</value>	The second se
<valuetype>Length</valuetype>	
<parameter></parameter>	
<name>CorbelTop</name>	
<text>Oberkante Konsole</text>	
<value>4000.</value>	
<valuetype>Length</valuetype>	
	ළ ණ 🖗 Close 🕕
	Set the to point Δx 0.000 Δy 0.000 Δz 0.000 🕅 🛆 🙏 🎢 🛱 0.010 - 🔚 -
× ×	
s	Press F11 Drawing type: Scale definition Scale: 1:100 Length: m Angle: 0.000 deg %: 1d

Figure 8: comparison between menu in Allplan and XML structure in notepad of the .pyp file

#### Editing a .pyp file

Files in .pyp format make use of the XML standard. This means that they can be easily edited. In XML we can make multiple objects and assign values to them. We can also put multiple objects within another object. This is called nesting. Every object (or section) is declared between <object\_name> and always ends with </object\_name> (notice the forward slash). Every PythonPart in Allplan has the following structure:

#### Descriptive line

The descriptive line contains the version of XML used and the font format (standard is UTF-8)

#### Element statement

Every PythonPart in Allplan will (besides the descriptive line) be edited within an <Element> </Element> statement.

#### Script statement

Within the Element statement, the most important to declare is the adjacent script that is connected to the .pyp file currently in. Allplan will, upon execution of the script, look for this file in the *"PythonpartsScripts"* folder. It is important that the syntax of the name is correct. Nothing will happen once you declare a wrong filename.

<SCRIPT>

<NAME>COLUMN1.PY</NAME> <GEOMETRYEXPAND>0</GEOMETRYEXPAND> </SCRIPT>

The script section contains of two parameters. The first parameter, **name**, refers the the location where the .py file is saved as stated above. Absolute paths, nor relative paths are possible. The location should always be in the destined folder within in Allplan. The second parameter, **GeometryExpand**, is very important. GeometryExpand defines if Allplan should let the Pythonpart react on other objects within its interface. A fine example of this is reinforcement that needs to be placed in a wall around a window. You want the reinforcement to react to the surrounding hole in the wall. Then you will have to set parameter GeometryExpand to 1.

Later on we will see that only setting this value to 1 does not suffice. An extra definition in the script's body needs to be defined.

As a general recommendation: we do recommend that you first develop your .pip files and define all the parameters. Your code in Python has to reference to these parameters. Things go a lot easier when parameters are created before starting on the code. It keeps things structured.

# Page statement

Page is a placeholder for the different tabs within an Allplan menu (cfr. to SmartParts). It works the same way as an Element statement. Whenever you want to close the page and start a new one, start a new page and close the previous one with </page>. Every page has two types of fields. A page has a name and a parameter section.

# Parameter statement of a page

One parameter section within a page might look like this:

<Parameter> <Name>ColumnWidth</Name> <Text>Breite</Text> <Value>600.</Value>

```
<VALUETYPE>LENGTH</VALUETYPE>
</Parameter>
```

In this example, the width of a column is defined. The name, text and value is not of utter importance and can be defined by the developer. The **valuetype** on the other hand is very important. The following types are recognized:

VALUETYPE	GENERATED INPUT FIELD IN ALLPLAN
LENGTH	ENTER A DISTANCE FIELD
LAYER	SELECT A SUBLAYER FROM DROP DOWN MENU
PEN	SELECT A PEN THICKNESS FROM DROP DOWN MENU
СНЕСКВОХ	DISPLAYS A CHECKBOX (VALUE 0 OR 1)
COLOR	SELECT A COLOR (ONLY ALLPLAN COLORS)
REINFBARDIAMETER	SELECT A DIAMETER FROM DROP DOWN MENU
INT	ENTER AN INTEGER (ONLY WHOLE NUMBERS)
DOUBLE	ENTER A DOUBLE (COMMA ALLOWED)
EXPANDERSTART / EXPANDEREND	ADD EXPANDERS TO CERTAIN PARAMETERS

#### Using the expander

The expander allows for showing certain parameters or omitting them in a drop down fashion. The expander code works as follows:

<PARAMETER>

<NAME>EXPANDER1</NAME> <TEXT>EXPANDER1</TEXT> <VALUE>TRUE</VALUE>

<VALUETYPE>EXPANDERSTART</VALUETYPE>

</PARAMETER>

... (PARAMETERS)

<PARAMETER>

<VALUETYPE>EXPANDEREND</VALUETYPE>

#### </PARAMETER>

The expander is firstly defined as a parameter. The value of the expander defines if it should be open upon start or closed. Ending a certain section of the expander works by defining a new parameter with valuetype "ExpanderEnd".

#### Adding an image

Adding an image is as of today not yet implemented (Allplan 2016.1.0)

#### Working with reinforcement

After closing the final page on your *.pip* file with a </page> statement, you can add reinforcement when needed. Reinforcement sections let you define placements for bars with certain parameters. Be warned that declaring a reinforcement shape does not automatically create an entry in a menu of the PythonPart to enable and disabling it. A parameter type checkbox should be created and should enable and disable the reinforcement shape in the script. Remember that the *.pyp* script absolutely has no intelligence. Any connection between scripts MUST be defined in the python script. One exception to this rule is the use of referencing to already existing parameters. One example is shown below where, in the menu, a "ConcreteCover\_reference" parameter is shown. In the reinforcement object it is possible to refer to this parameter through the *.pyp* file alone.

<REINFORCEMENT>

<SHAPENAME>USER SELECTABLE PART OF REINFORCEMENT</SHAPENAME> <ID>0</ID>

<DIAMETER>8</DIAMETER>

<DISTANCE>100</DISTANCE>

<CONCRETECOVERSHAPE>CONCRETECOVER\_REFERENCE</CONCRETECOVERSHAPE>

<PLACEMENTCOVERLEFT>CONCRETECOVER\_REFERENCE </PLACEMENTCOVERLEFT>

<PLACEMENTCOVERRIGHT>CONCRETECOVER\_REFERENCE </PLACEMENTCOVERRIGHT>

<CONDITION>SHAPE1</CONDITION>

</REINFORCEMENT>

VALUETYPE	DESCRIPTION
SHAPENAME	NAME OF THE PLACEMENT
ID	ID NUMBER
DIAMETER	DIAMETER OF THE BARS
DISTANCE	DISTANCE BETWEEN THE BARS IN THE PLACEMENT
CONCRETECOVERSHAPE	COVER OF THE BARS IN RELATION TO THE SHAPE IN PI
CONCRETECOVERLEFT	END COVER LEFT OF BARS
CONCRETECOVERRIGHT	END COVER RIGHT OF BARS
CONDITION	UNKNOWN PARAMETER (CONDITION=SHAPENAME)

# Externally editing parameters for Allplan

Although it is possible to change the parameters inside Allplan through the menu that shows when dragging them inside the program, a much more interesting approach would be to externally access and edit the files.

An example could be a website where users could define their own preferences for a beam. On the website, a plain XML file is generated with PYP code. This file can now easily be dragged in Allplan (providing the script is functional) without the user having anything more to do than extracting the file from the website and opening the finished beam in Allplan.

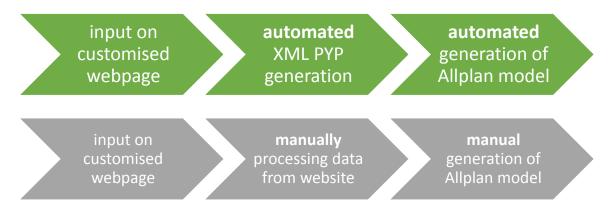


Figure 9: comparing the old and new workflow of Allplan with PythonParts

Currently, a test version of this XML PYP editor/generator is under development @SCIA Herk-De-Stad.

# Advanced Editing of PythonParts in Visual Studio

# Setting up the workspace

In order to use Python in Visual Studio (which we already covered in chapter one) and more precise, use the libraries provided by Nemetschek for development, we need to set up the Visual studio environment.

# Allplan libraries

The Allplan libraries which we need for development can be found in the PRG folder in the Program Files. Libraries for Python always have *.pyd* file extension. Remember the path to the files.

🛃 📙 🖛   Prg				
File Home Share	View			
> -> 🛧 📙 -> Th	is PC > Local Disk (C:) > Program Files > A	Allplan > 2016_Verificatio	on 2016 > Prg >	
A Quick access	Name	Date modified	Туре	Size
Desktop 🖈	Ec2n_n32.deu	03/10/2015 09:02	DEU File	7 KB
	📀 Setup	09/07/2015 15:26	lcon	27 KB
Uownloads 🖈	NemAll_Python_Elements.pyd	14/10/2015 09:36	PYD File	197 KB
🔋 Documents 🖈	NemAll_Python_Geometry.pyd	13/10/2015 23:41	PYD File	10,051 KB
Pictures 🖈	NemAll_Python_IFW_Input.pyd	13/10/2015 23:42	PYD File	271 KB
OneDrive	NemAll_Python_Palette.pyd	13/10/2015 16:20	PYD File	109 KB
oneonie	NemAll_Python_Reinforcement.pyd	08/10/2015 23:55	PYD File	606 KB
This PC	NemAll_Python_Utility.pyd	08/10/2015 23:40	PYD File	397 KB
Network	7z-license	09/07/2015 15:13	Text Document	2 KB
NEWOR	avny evt th	03/10/2015 04:59	TLR File	56 KR

Figure 10: locating the Allplan PYD libraries

LIBRARIES	DESCRIPTION
Elements	?
GEOMETRY	CONTAINS ALL GEOMETRY FUNCTIONS AND OBJECTS
IFW_INPUT	?
PALETTE	?
REINFORCEMENT	CONTAINS ALL REINFORCEMENT FUNCTIONS AND OBJ
UTILITY	GENERAL UTILITIES

# Python Project in Visual Studio

You can either create a new Python project manually in Visual Studio, or use the template provided by your Allplan reseller.

When using the template, open the *.pyproj* file in a text editor and change the paths of the <reference include="..."> and all other paths to the appropriate path in the library. Open the template afterwards, Visual Studio will start with the necessary libraries already included.

When starting from scratch, create a new Visual Studio project and select Python Application from the list. Afterwards you need to add the libraries yourself. Therefore click right on the references in the right pane and select "add reference". Now you can also browse for the libraries. With shift-click you can select multiple libraries at the same time.

	-	Solution Explorer		<b>-</b> ₽ ×
		C O ∰ To -	2 🗇 🗿 🕨 🗕	
	÷	Search Solution Explo	orer (Ctrl+\$)	- م <u>ر</u>
		A PY PythonApp	nApplication1' (1 project) l <b>ication1</b> Environments	
		🛤 Referenc	es	
	Add Reference		aths	
苗	Manage NuGet I	Packages	pplication1.py	
	Scope to This			
	New Solution Ex	plorer View		

Figure 11: adding a new library to the project

New Project					? ×
▷ Recent		.NET Fr	amework 4.5 - Sort by: Default	• # E	Search Installed Templates (Ctrl+E)
▲ Installed		<b>∑</b> J	From Existing Python code	Python	Type: Python
<ul> <li>▲ Templates</li> <li>▲ Visual Basic</li> <li>▶ Store Apps</li> </ul>		Python Application	Python	A project for creating a command-line application	
Windows Do	esktop	$\bigcirc$	Azure Cloud Service	Python	
Cloud Reporting		Ű	Web Project	Python	
Silverlight Test		Ű	Bottle Web Project	Python	
WCF Workflow		Ű	Django Web Project	Python	La contraction of the second s
▷ Visual C#▷ Visual C++		Ű	Flask Web Project	Python	10
<ul> <li>Visual F#</li> <li>SQL Server</li> </ul>		Ű	Flask/Jade Web Project	Python	
PowerShell ▷ JavaScript		<b>∑</b>	IronPython Application	Python	
▲ Python Web	_	PY	IronPython WPF Application	Python	
▷ Online		₹ <sup>PY</sup>	IronPython Silverlight Web Page	Python	~
			Click here to go online and find templates.		
Name:	PythonApplicatio	on1			
Location:	c:\users\bertvo\documents\visual studio 2013\Projects -		Browse		
Solution name: PythonApplication1			Create directory for solution		
					OK Cancel

*Figure 12: selecting Python application from the Python tab* 

When going to project > properties, you can select the interpreter to be used. When no interpreter is available, please refer to the installation chapter. An interpreter is the vital component that will execute and compile your code and check for errors. You can also define a startup application. In most cases you are only going to have one script in a project. So the name of that script should already be in the list.

Debug				
Publish	Application			
	Startup File	PythonApplication1.py		
	Working Directory			
	Windows Application			
	Interpreter:	(Use global default)		
		- (Use global default) Python 64-bit 3.4		
		rython of-bit 5.4		

Figure 13: selecting an interpreter in the properties pane

Double clicking on the script in the right pane should open the script and show an absolutely empty screen with a cursor

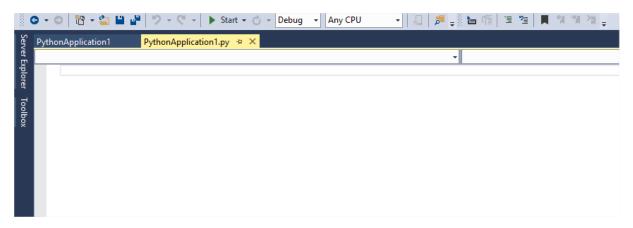


Figure 14: the empty script

In order to test if everything is working, type <print ("hello")> (without <>). You should get an output window showing you hello after clicking on the green little arrow in the top screen.

- Start -
Figure 15: run application
Process: [4100] python.exe     ▼ LE Lifecycle Events ▼ Inread:
PythonApplication1 PythonApplication1.py + ×
<pre>print("hello")</pre>

Figure 16: application running and well - Visual Studio has been set up correctly

# Analyzing an example script

# Generating PythonDoc

To help you get going, it might be useful to generate documentation of all the definitions in the Allplan libraries. This can be done through the test projects included in your installation.

When using the project template, these will already be in the list with python scripts. When you created a project from scratch, you will have to drag the file from the Examples/PythonParts folder into your project's solution explorer. It will be added to the project and automatically use your imported libraries.

Import the "createHTMLDocumentation.py" file in your project. A little bit down in the script you will see the following:

 $DRIVE\_LETTR = 'X: \backslash \backslash'$ 

Change X to the drive on which you installed Allplan and run the script. Documentation is now written to *ETC/PythonPartsScripts/Docu*. The documentation consists of html files that are easy to read.

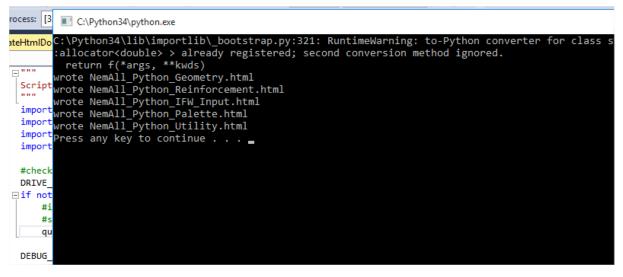
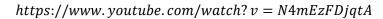


Figure 17: successfully generated pythonDoc

# Before we start

In this tutorial series, we do expect some basic knowledge of programming. If you are unfamiliar with how Python works or coding in general, we refer to the following tutorial on YouTube in English. In order to fully understand the next paragraphs, you should be familiar with concepts like: definitions, classes, inheritance, objects, instances and general python syntax.



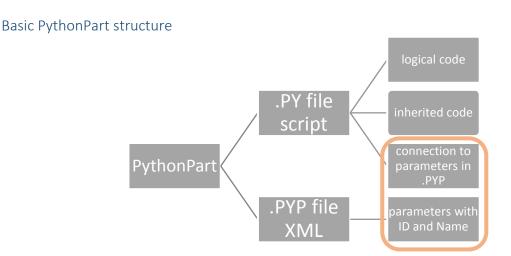


Figure 18: the basic guideline scheme every PythonPart needs to follow

A PythonPart mainly consists of two files, the PYP and the PY file. As discussed above, the PYP file contains parameters for menu creation and reference in XML format. The PYP file can also be used as an indirect modifier of the PythonPart through a basic XML editor. More complex changes and adaptations to Objects in Allplan (E.g. reacting to existing object in the Allplan object space) need to be done with the use of real Python Scripting. This is where the PYP file comes in.

Within the PY script file, a connection is laid with the parameters defined in the PYP parameter file. If this connection has not been executed properly, or some objects are not referenced to, nothing will happen upon executing the script in Allplan.

Furthermore, the PY script file "inherits" a few definitions. Upon dragging the file into Allplan, the program will start looking for these definitions and try to execute them. The output should ALWAYS be the same. There are three types of main definitions in Allplan script language. They will be discussed later on.

# Defining inherited methods

Although, Python is unable to inherit certain properties from classes (in comparison to Java for example), inheritance in Python can be seen as the knowledge that the following three Python Definitions will always be accessed by the Allplan Interpreter. It is therefore important to define them properly.

DEFINITION NAME	USAGE ALLPLAN	ASSOCIATED PARAMETER	OUTPUT
CREATE_ELEMENT():	ELEMENT GEOMETRY	NONE	GEOMETRY
MOVE_HANDLE():	ELEMENT HANDLES	NONE	GEOMETRY,
			HANDLES
EXPAND_CREATE_ELEMENT():	EXPANDED GEOMETRY	GEOMETRYEXPAND (0/1)	BOOLEAN CHECK,
	REACTS TO OBJECTS		POINT, GEOMETRY

Figure 19: different definitions that should be used in Python for Allplan

"Expand Create Element" makes use of the GeometryExpand parameter. If this parameter is set to zero, the definition does not need to be executed and defined. A basic Python script structure for a beam would be like below.

(IMPORT STATEMENTS OMITTED)

DEF CREATE\_ELEMENT(BUILD\_ELE, DOC):

ELEMENT = BEAM(DOC) //REFERENCE TO THE BEAM CLASS

```
( ...)
```

RETURN ELEMENT.CREATE(BUILD\_ELE) //RETURNS GEOMETRY

DEF MOVE\_HANDLE(BUILD\_ELE, HANDLE, HANDLE\_PROP, INPUT\_PNT, DOC):

```
( ...)
RETURN ELEMENT.CREATE(BUILD_ELE) //RETURNS GEOMETRY
```

#### DEF EXPAND\_CREATE\_ELEMENT(BUILD\_ELE, EXPAND\_UTIL, REF\_PNT, VIEW\_PROJ, DOC):

```
ELEMENT = BEAM(DOC) //REFERENCE TO THE BEAM CLASS
( ...)
```

*RETURN (TRUE, MODEL\_PNT, ELEMENT.CREATE(BUILD\_ELE, BEAM\_LENGTH, BEAM\_HEIGHT))* 

# CLASS BEAM():

```
DEF __INIT__(SELF, DOC): //INITIALIZE THE BEAM > THE REFERENCED DOC IS ALWAYS PASSED THROUGH! (...)
```

We can see that the following structure is maintained: Object data gets created in in the Object class (below), this data gets handled to Allplan through the basic definitions. If only defining a simple beam, *create\_element* and *move\_handle* are enough. When creating reinforcements that react to

the environment, expand\_create element is necessary. (*create\_element* is obsolete as one of the two gets selected because of the *expandGeometry* parameter)

When handling reinforcement, this should also be passed through the *create\_element* method together with the other geometry. Allplan will handle the recognition of different elements.

# Referring to parameters from the python script

Assigning parameters from within the Python script is fairly easy once you have created the correct parameters in the PYP file. An example will clarify this.

When I define a parameter with <*name>Beam\_Width*</*name>* and <*value>200*</*value>* I can refer to this parameter in the Python script as follows:

```
class beam():
(...)
Def create_geometry(self, build_ele): //created a definition for simplification, not necessary
    beam_width = build_ele.beamWidth.value
    (...)
    column = AllplanGeo.Polyhedron3D.CreateCuboid(beam_width, (...) )
    (...)
```

As you can see, the assignment is fairly easy. In this case, *Beam\_width* in the Python script will be 200, and this value can, afterwards, be used to create the column with another function (in this case *createCuboid*).

All parameters, in order to use them properly, need to be referred to. The only parameters that are handled by Allplan are *ExpanderStart, ExpanderEnd* and *GeometryExpand*.

# Import statements and python philosophy

Every script, as described above, should start with the import statements and the definitions that Allplan will use. In these import statements, it is of course possible to import your own scripts. In case you want to use an external interface to change some object properties, or you want to write an import application for Allplan, Python can be easily used to create a few python scripts that handle these. A major advantage of using Python is data conversion. It is now possible to create your own data converters. You handle the conversion, you control the data loss and conservation, and you are in charge. This is contrasting to how import was handled before in Allplan.

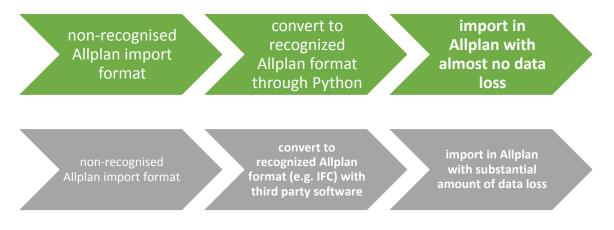


Figure 20: advantages of Python in the Allplan import Workflow

# Simple column script example

We will discuss the creation of a simple column in Allplan. The column has a little reinforcement as an example and is adjustable.

In the Allplan PythonPart example scripts, a column1.py script can be found. This will be used as a first basic example. Besides the basic imports from the *Allpan NemAll* libraries, this simple class also uses predefined scripts for easier object handling. We will keep the description and explanation limited to the functions used in the column script.

TODO