Revit API – An Introduction : London Revit User Group Presentation : 16.03.2011 : Notes

Ritchie Jackson

01

The aim is to provide novice or hobbyist programmers with a starting point to the API. The focus will be on the generation of form rather than project analysis and data extraction.

02

Target Audience : Familiarity with Revit is an obvious necessity. Novice programmers include those with Java experience as the crossover to C# is relatively easy.

Interface : When starting to use the API, experimenting with VSTA Macros is more straightforward than creating External Command or Application Add-Ins

Focus : Much of the underlying geometry is common to a variety of component types and familiarity of the former is essential for building projects. Once the building blocks are mastered then the appropriate component type – Mass, Generic, System and so forth – can be chosen.

Examples : The code covers the generation of a Line, Extrusion, Extrusion with Cut-Out and Iteration using the ‘for-loop’ – all from within a Generic Model family document. Generic Model Extrusion and System Floor are compared to highlight their code similarities. A few projects are discussed where the API has proved to be useful.

Code Appendix : Only the ‘Line’ example has been covered in the presentation as the accompanying code for the others has been extensively commented.

03

API use and appropriateness : This will depend on many factors – specific project deadlines, team expertise, reusability of macros in future projects. Initially, however, time will be spent in coding items which could be done more quickly by standard UI methods. This is an unavoidable consequence of the learning process.

04

Examples with C# Code : An overview of the introductory macros provided.

05

Workflow Comparison : The two forms are identical in appearance but have inherently different characteristics due to their family types. However, most of the underlying code (Points, Lines, Arcs and CurveArrays) is shared. Once the latter is mastered it is easy to generate the appropriate form for the task at hand. A Generic Model Family is a good learning starting point.

06

Project API Examples : The usefulness of the API was tested in four disparate projects. It proved to be invaluable in the *High-Rise* example due to the complexity of dependent components – floors, support beams and façade panels. A variety of floor-plate configurations could be quickly generated, placed at a few specific levels and used to control the overall façade set-out. The *Roller Coaster Reception* was intended as a playful excursion to investigate the potential for hooking visible geometry onto a single spline armature. Whilst something similar could be achieved conventionally in a Conceptual Mass family, adding complexity by this method often leads to problems of selection and association which are difficult to resolve. Similarly, with the *Truss* – once the initial code in the API is debugged it is relatively straightforward to increase complexity without ‘breaking’ the original alignments and dependencies. The *Amphitheatre* used the API for preliminary setting-out only.

07

Macro : Basic_01_Line : The workflow is shown in pseudo-code form and should not be copied / pasted into the macro editor. The '7 lines of code' refers to the Line-specific elements – the macro itself requires more method calls which are covered in the functioning supplied code. A Generic Model family document is used to create the element.

08

Macro : Basic_02_Extrusion : Comments as **07**.

09

Macro : Basic_03_Extrusion : Comments as **07**.

10

Macro : Basic_04_BoardWalk : Comments as **07**. A Hermite Spline which interpolates five control points is set-out on the Ref.Level plane. This *can* be displayed (useful for debugging) but this is not necessary. A series of unbound lines parallel to the y-Axis are generated through an iterative 'for-loop' at regular intervals (plank-width plus plank-spacing) and their intersection points with the spline determine the end-point lengths of each plank. The height of each individual plank is set as a proportion of its length – obviously many alternatives are possible. Code from the previous example is modified and encapsulated in the 'drawPlank' function which is called from within the 'for-loop' to generate individual planks which are topologically similar but dimensionally different. The power of the API starts to be utilised – as the example is relatively simple it could be achieved through the conventional user-interface. However, when learning to use the API one has to start with small steps – the purpose of this presentation.

11

High-Rise : Scope : Both the Podium (with the exception of glazing panels) and Core were created in a conventional project environment. The Tower made extensive use of the API as noted on the following page. The Apex was constructed manually with the exception of the glazing, where code from the Tower was *re-used* and *adapted* to suit – an example of leveraging pre-existing routines. The floor-plate support beams were also created via code, with their lengths determined by the intersections of the floor outer-profiles.

12

High-Rise : Workflow : Each floor outer-profile consists of 24 control points generating 4 interpolated splines – whilst the plans shown are rotationally symmetric, they do not need to be and can take on more organic, asymmetric shapes. In the example shown, four different floor profiles are placed at nine levels to control the overall form of the façade. This arrangement can be infinitely varied to suit particular design constraints. Twenty-four mullion splines then interpolate the floor-plate control points. These mullions are then intersected radially on a floor-by-floor basis to find the set-outs for the intermediary floor-plates as well as the glazing transoms. From this armature, the Floor Plates, Beams and Façade Panels can be constructed. In the test case these consisted of Extrusions and Blends – however the same geometry can be fed into System Floor- and Curtain Wall elements. There are twelve pre-set materials for the glazing panels which are automatically assigned in a random fashion.

13

Amphitheatre : Set-outs consisting of Lines and Arcs were coded and displayed as ModelCurves and a number of configurations were tested to suit a particular design brief before the structure and seats were added via conventional means. The formal elements could also have been created via programming but for a one-off design this was not deemed to be worth the effort.

14

Truss : A clear instance of a form which could be easily constructed using conventional means. The API was used as a learning exercise. However, as noted for the Amphitheatre, it was found that adding complexity (number of components, level of detail) can often be more easily accomplished through programming.

15

Roller-Coaster Reception : Primarily an exercise to investigate the API-accessible properties of a Hermite Spline from within a Generic Model family. The spline *interpolates* the control points – an option not available in the UI in the 2011 version (hopefully addressed for 2012?). The underlying 3D spline can be *visualised* in a Conceptual Mass family (or external package like 'Rhino') for sketch purposes and the control points exported.

16

Roller-Coaster Reception : Workflow : The control points are imported into the Macro and the tangent vectors are determined at regular intervals along the spline from the 'BasisX' component of the 'ComputeDerivatives' function. These then provide the normals for the set-out planes at these intervals on which equilateral triangle bracing frames are constructed. The latter are simply scaled to taper towards the spline's upper end-point. Each frame triple is used as the set-out for a connecting arc to provide a means of rationalising a spline of continuously changing curvature into a series of fabricatable components with 2D true-arc generators. The number of intervals chosen is solely dependent on achieving *visual* smoothness of form. That is, each arc junction should *appear* to be being co-tangent. The rafters spring from the bracing frames and their lengths are controlled by a sine function.

17-26

Appendix : Module and Macro : 'Quick Start' and 'Starting from Scratch'. The code is provided 'as-is' and kept to the bare minimum to aid the learning process. Users should enhance it with their own error-checking and object-oriented methods. The full macro code for 'Basic_01_Line' appears on Page 25.