Simio
Forward Thinking

Simio API Note: Interfacing with Visio

History

Creation: 06 Aug 2018

Update: 23 Feb 2023

Simio
Forward Thinking

Contents

Simio
Forward Thinking

A Note About this Version

This is a beta version to present the concepts that are being employed to permit drawings to be exchanged with Simio. As explained below, this technique introduces the **S**imio **D**rawing E**x**change (SDX) conventions that will be used to convert drawings to a format (a DataSet using SDX conventions) that Simio understands.

This release contains the following changes from the beta version:

1. Many minor bugs were fixed
2. The ability to reference object subclasses
3. The ability to include Simio properties from Visio
4. The ability to set the Simio name from Visio
5. Prevents importing a model on top of another model
6. The ability to automatically scale objects to normal Simio sizes.

To accommodate these, the SDX dataset went through some major revisions, including the addition of two more tables. The SDX dataset version is 18.09.07

Simio
Forward Thinking

Overview

A Microsoft Visio document is a popular way to design and display visual information. This document describes how information created with Visio can be exchanged with Simio.

The simplest description of the process is this:

A Visio drawing is created (perhaps by using one the Simio Visio Stencils, such as SimioFacility.vssx). Once the diagram is created, the Visio Add-In is run. It prompts the user for (1) the Visio VSDX file and (2) a transform to locate the Visio drawing in the Simio location. This transform consists of an XYZ offset and scaling information, so that we know where to place the Visio objects and how big to make them. It then proceeds to creates the corresponding Simio Intelligent Objects (SIOs) and their links (and the link vertices).

Note: the Simio Visio Stencils are intentionally small and simple. This was done to allow drawings to fit more easily on a Visio page. Below the procedure is discussed to transform the drawing.

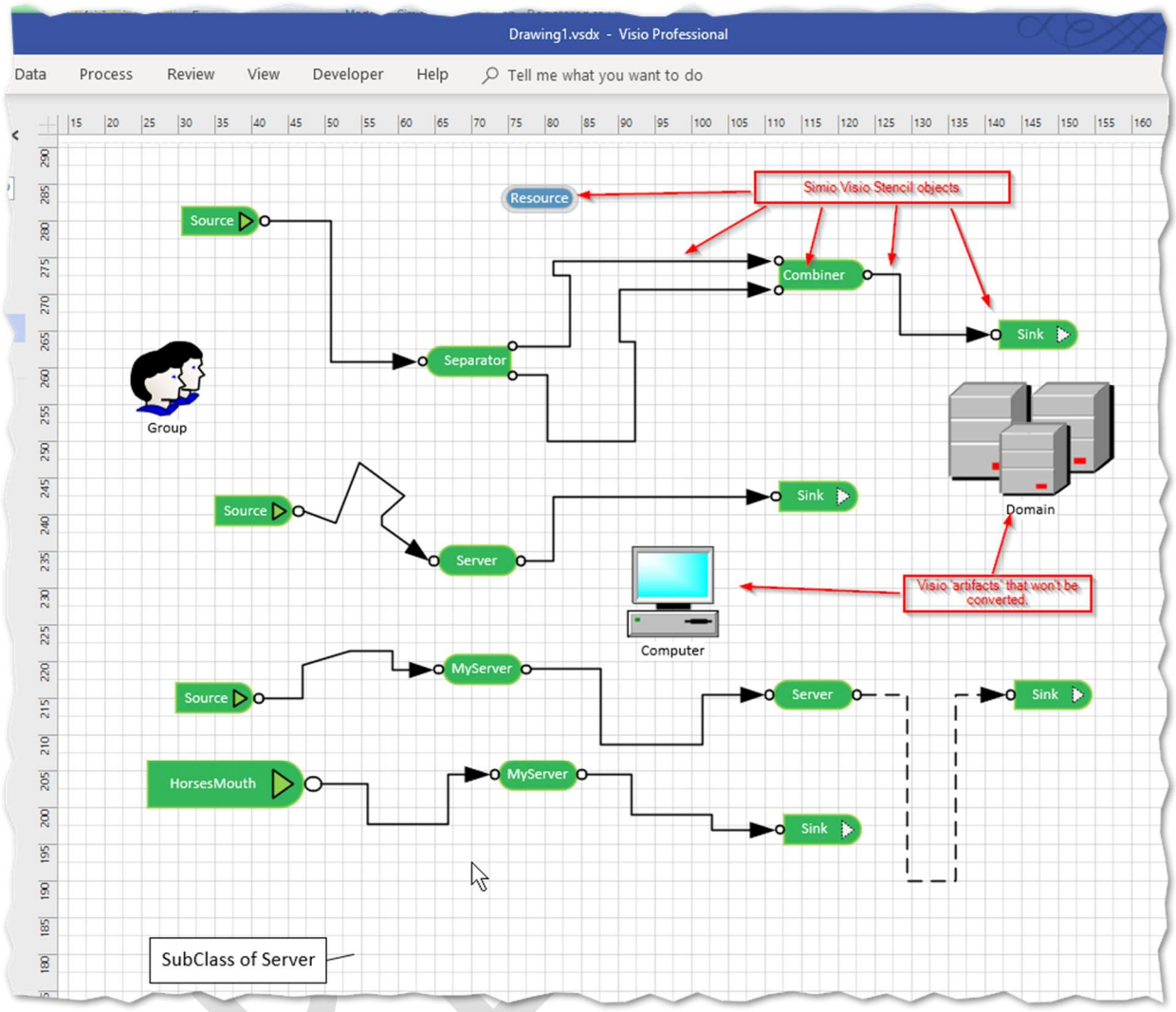


Figure 1 - Diagram Created in Visio using the Simio Facility Stencil

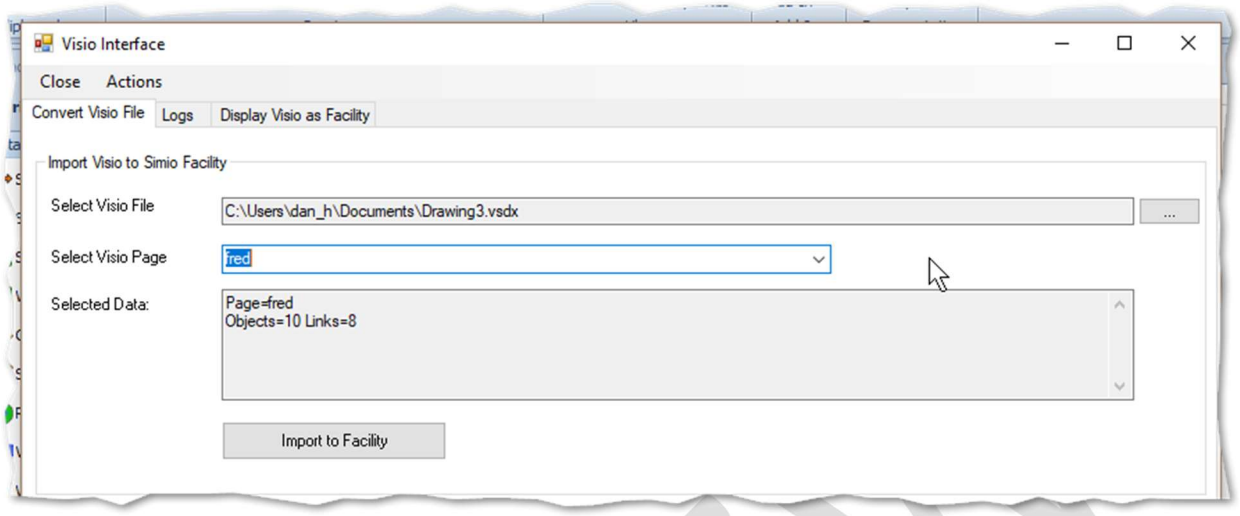


Figure 2 - Running the Add-In: Prompting for the Visio File

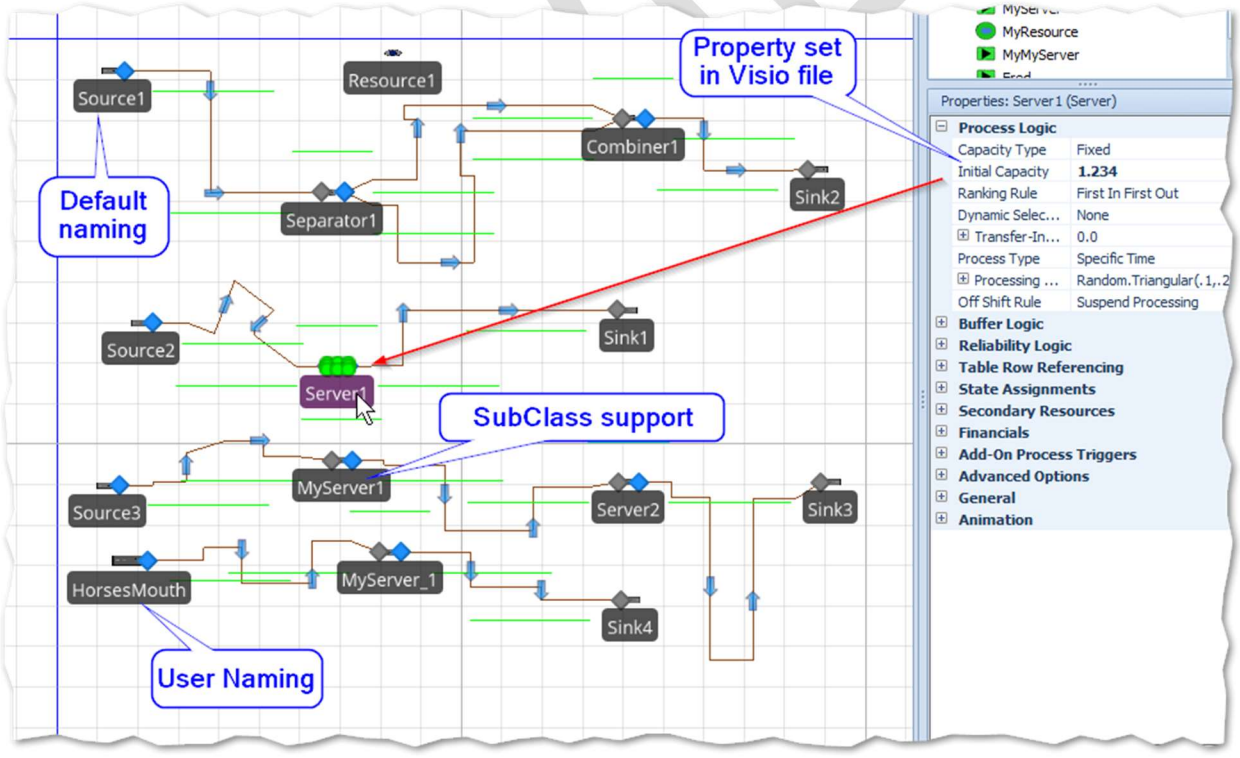


Figure 3 - Converted Diagram Running in Simio

Simio
Forward Thinking

Quick Start

On the Simio Forum, a Package file (VisioAddIn.zip) is provided.

A small example illustrates this process. Using Visio and the Visio "Simio Facility" stencil, create a Source-Server-Sink model, as shown:

Make sure the VisioAddIn is in your user Document >> SimioUserExtensions folder.

Run Simio, and once loaded select the Facility tab and then Project Home >> Select Add-In and then choose VisioAddIn.

A form will prompt you for a location and scale. Just accept the default and press Import.

The Visio model displays on the view, and you can press the Simio run button to begin the simulation.

Design Considerations

The Interface was built to have the following characteristics:

- It is consistent with other Simio-Interface standards, such as Excel-Import
- It can work with Visio drawings built with the Simio Stencils or with other Stencils if the user adds properties (Shape Data) to the shapes.
- It is consistent with Simio's Data-Driven model philosophy
- It does not require COM or Interops, and it works with .NET Core
- Simple usage: minimal programming
- It does not need an installation of Visio to work; it only requires the .VSDX file.
- It uses not require 3rd party commercial products
- It leverages standards, such as Open Package Convention (OPC), Linq, and XML as much as possible.
- It provides accurate logging to help locate problems.
- Well-documented samples are provided with it.
- It defines a standard and workflow that could be used for other visual products; for example, importing AutoCAD drawings.

Simio
Forward Thinking

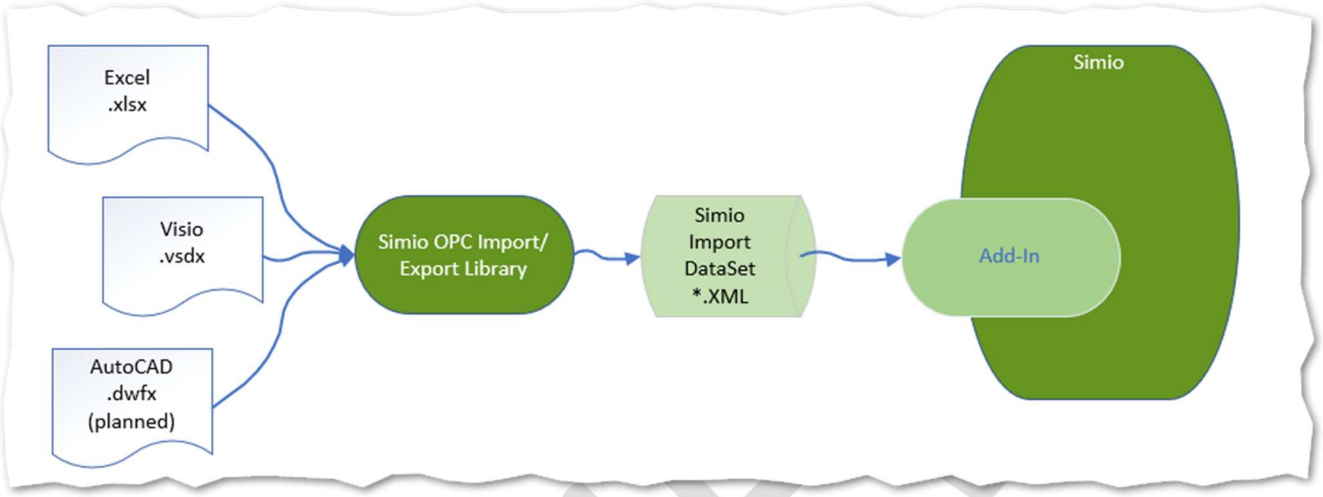
What's Included

Included with the Visio example is:

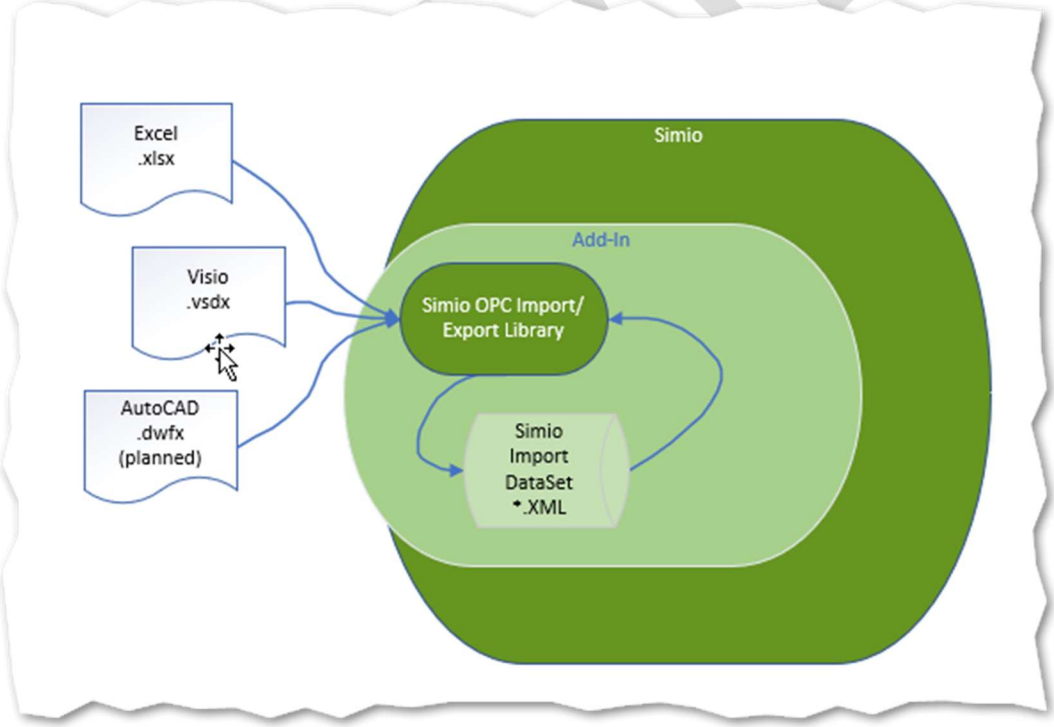
- All Source code (only the Add-In source is included now. All source will be eventually included)
- A Utility called SDX Harness that permits you to examine and preview your Visio document.
- Visio stencils for the Simio drawings (only the Facility stencil is included at this time).
- An example Visio VSDX file containing basic Simio Shapes from the SimioFacility Stencil.
- A library for generating a Simio DataSet from a Visio file.
- A library for using a Simio DataSet to create a Simio Facility
- A Design-Time Add-In that employs both libraries to convert a Visio file to a Simio Facility
- An accompanying 5 minute video that illustrates the process.

Simio
Forward Thinking

Based on these characteristics and input from Simio users and partners, the following architecture was derived:



Note that the Library can be used by external programs, but can also be called from within the Add-In, and could then be visualized like this:



Regardless, the process is still the same: A Simio-Visio-OPC Library is used to read the Visio file and creates a standard Simio Import DataSet. This DataSet contains seven Tables:

1. Properties to identify the DataSet type and contents
2. Objects to hold the Simio objects such as Source, Server, Sink, etc.
3. ObjectProperties, which hold Simio properties for Objects
4. Links to hold the connectors
5. LinkProperties, which hold Simio properties for Links
6. Vertices which hold information about how to draw the connectors
7. Artifacts that were found, but cannot be converted to Simio

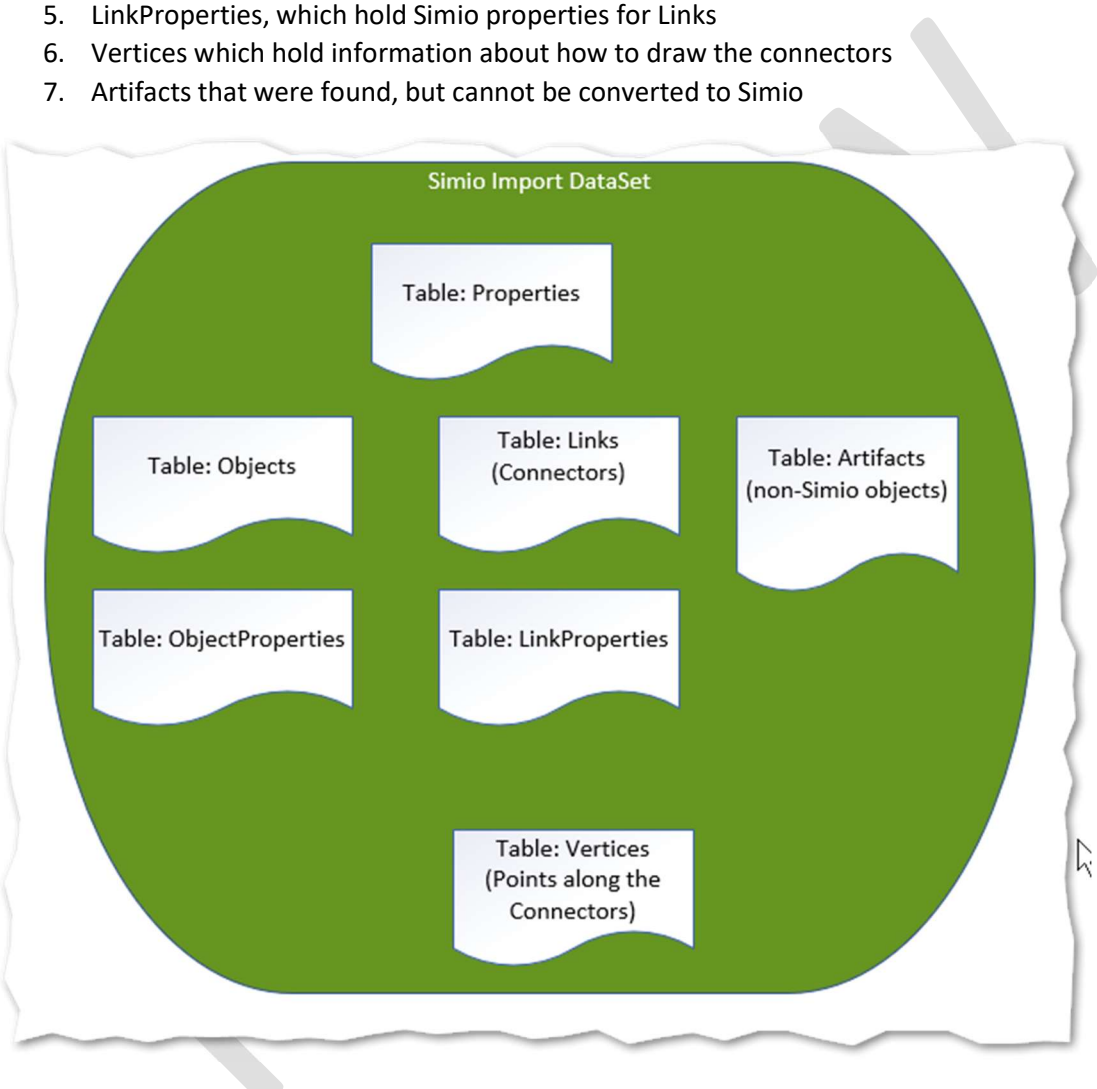


Figure 4 - DataTables in the SDX DataSet

There are three different types of DataSets corresponding to Simio views:

1. Facility View
2. Process View
3. Task View

In this release there are examples only for Visio, with Excel to be included before the final release. Other applications that employ the OPC Package file standard - such as AutoCAD (Autodesk) and Simulink (MathWorks) - are also candidates.

Also, this release only contains an example for the Simio Facility View. The Process and Task views are planned for a future release.

A Simple Example

Consider a Visio example that is to server as a basis for a Simio project. This is a worse-case example, since this drawing only contains pictures. That is, there is no inherent logic, only the rough shape and placement could potentially be used.

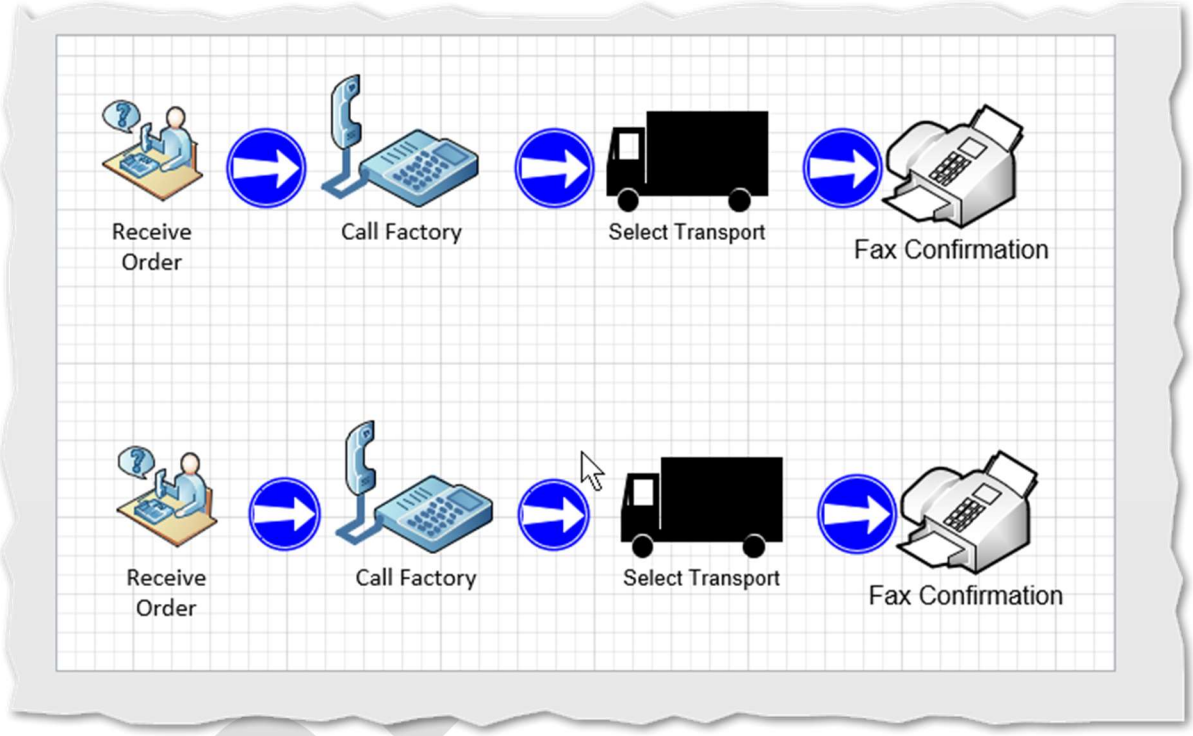


Figure 5 - Simple Example. No inherent logic.

When the SDX Harness is run, the Preview window displays the objects in gray, as they are not candidate Simio objects, as there are no Simio Shape-Data properties.

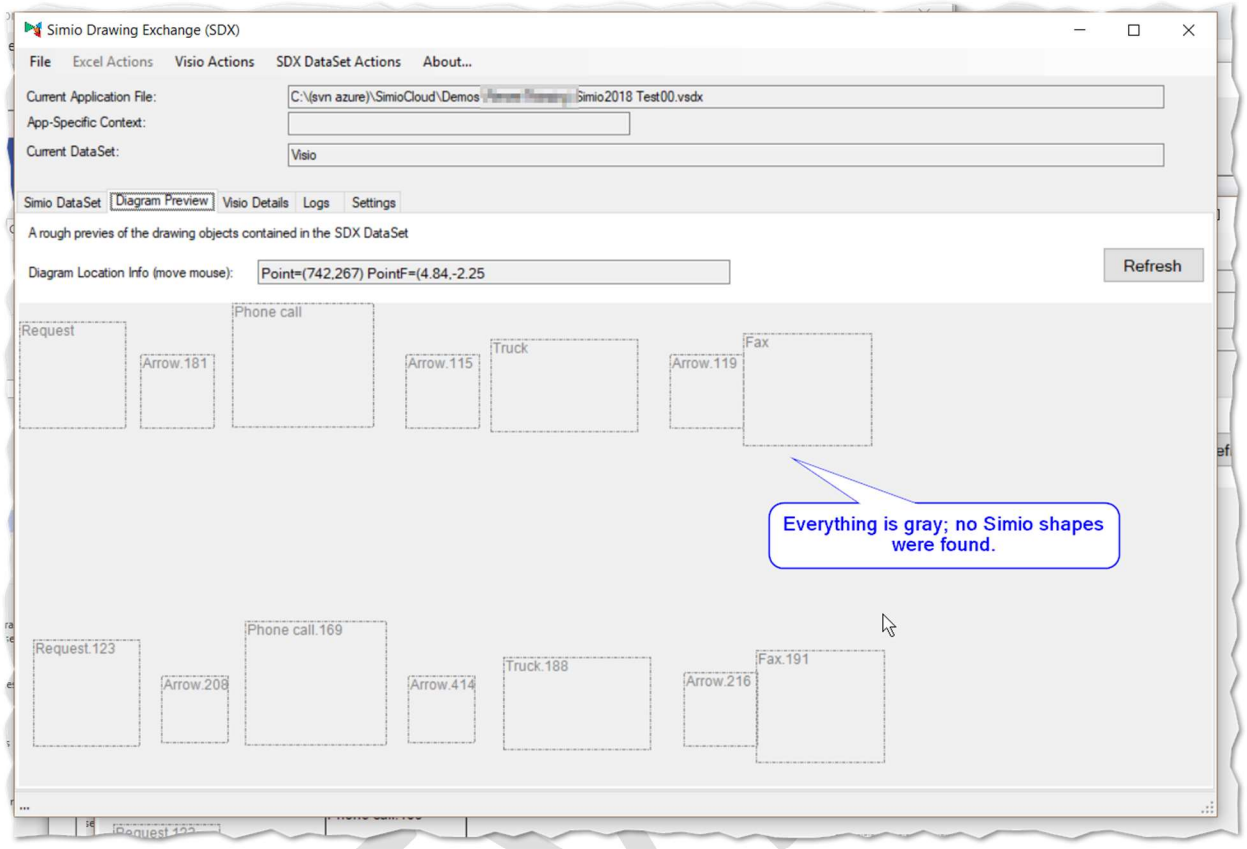


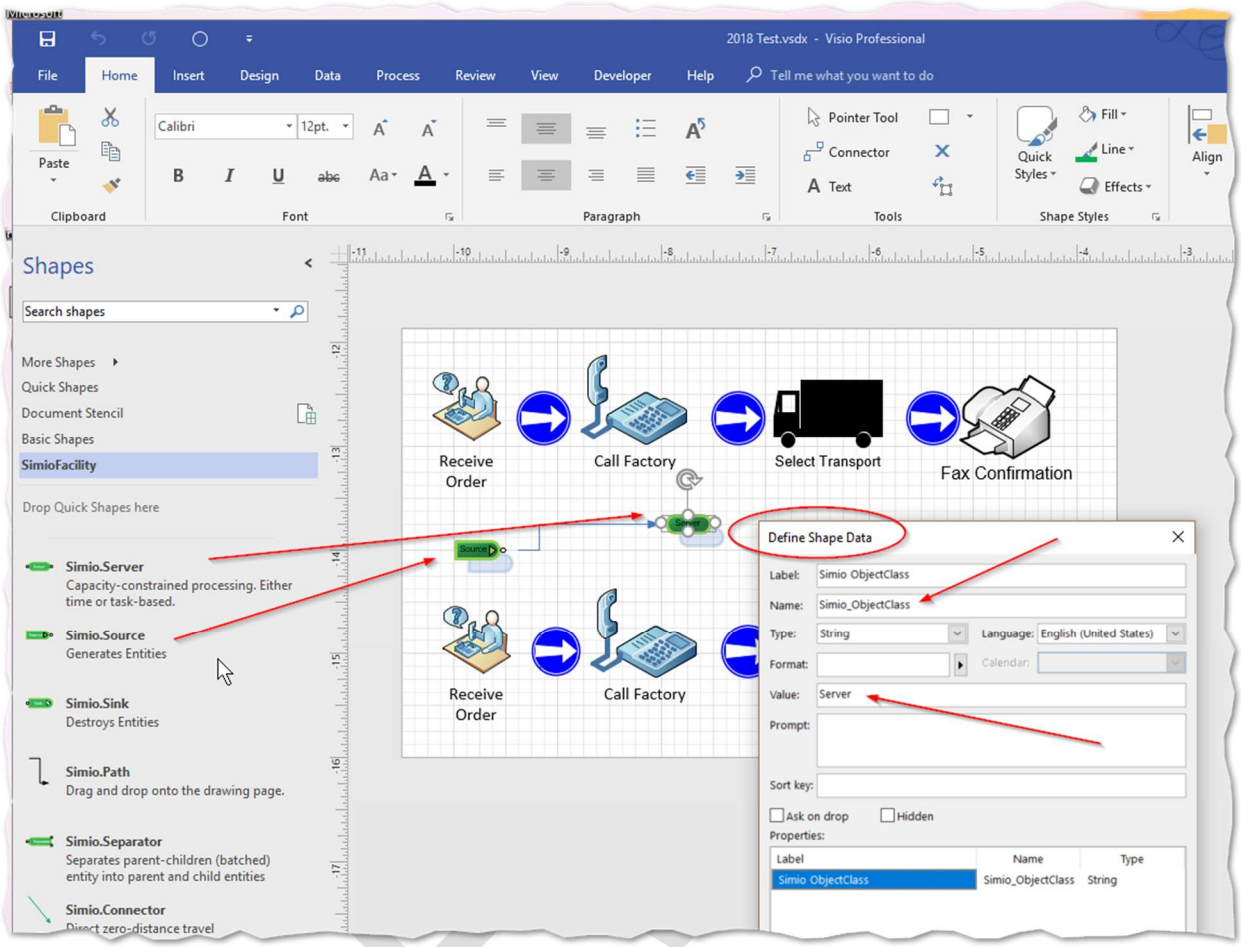
Figure 6 - Simio Import Preview

So, there are two possibilities for adding more information so that the Add-In can make Simio objects:

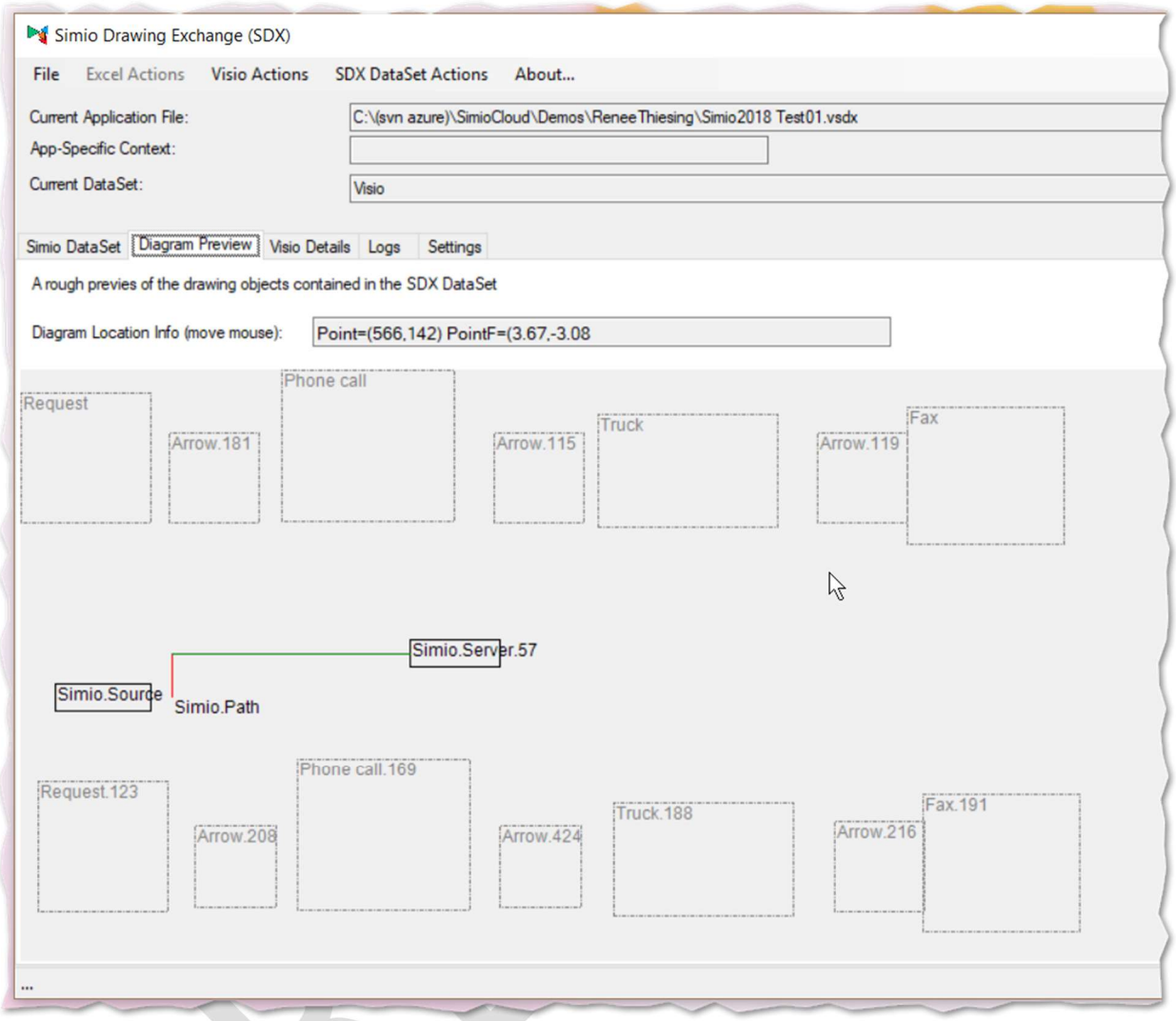
1. Use the Simio stencil, or
2. Add Shape-Data to make them Simio-able.

In the first case, we'll use the Simio stencil and add a simple source-server and import again to demonstrate the difference.

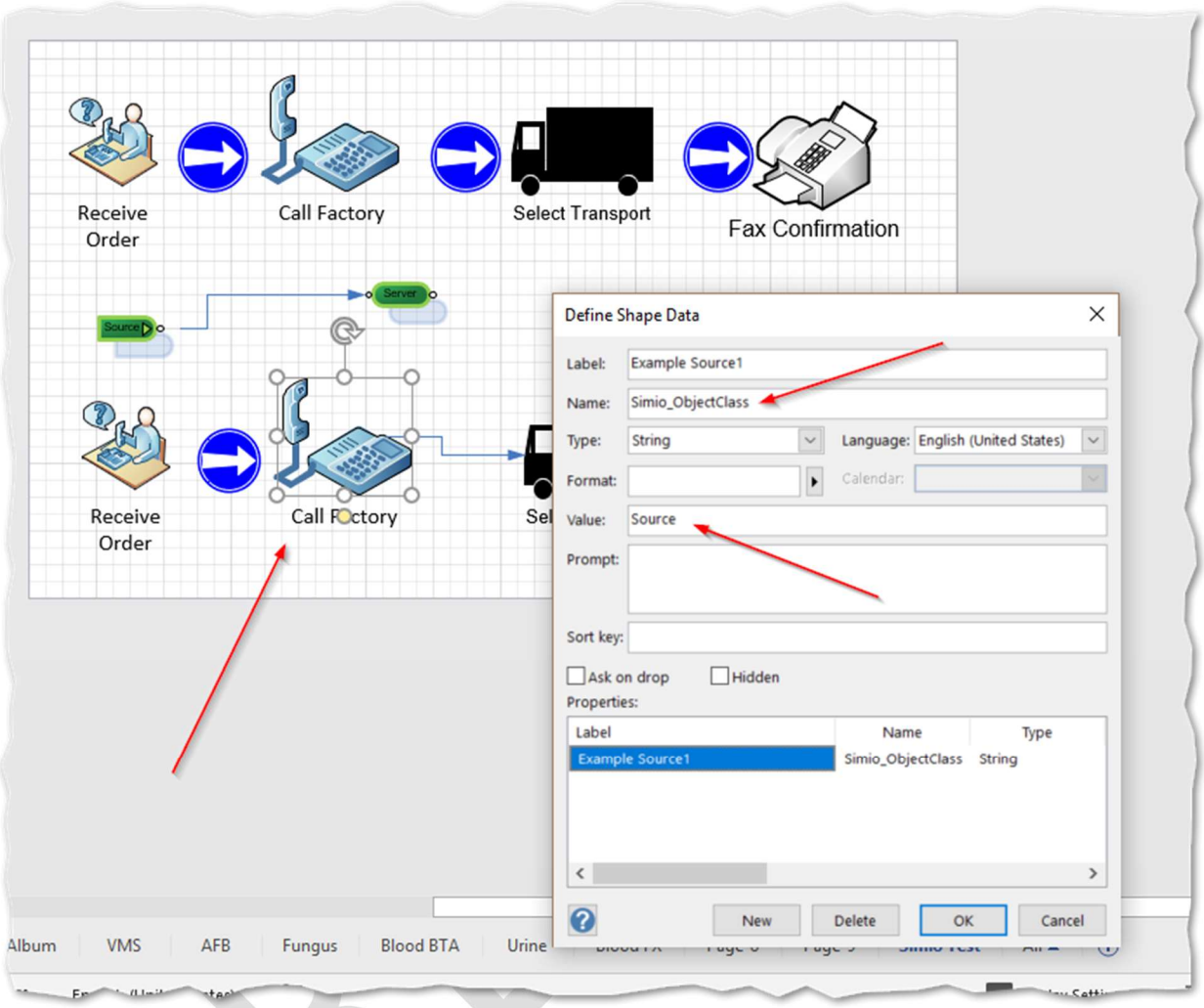
From the stencil, a Source and Server are added. An MRC (Mouse-Right-Click) on the server and selection of the "Define Shape Data" shows the values of the required Name (Simio_ObjectClass) and Value (Server).



When the Import is done, the Source and Server are now visible and highlighted.



Next, we'll alter the Shape-Data of "Call Factory" and "Select Transport" to indicate that they are Simio Source and Server, respectively. And then well add a Simio.Path between them:



When this is imported, the Previewer shows that they are now treated as Simio objects:

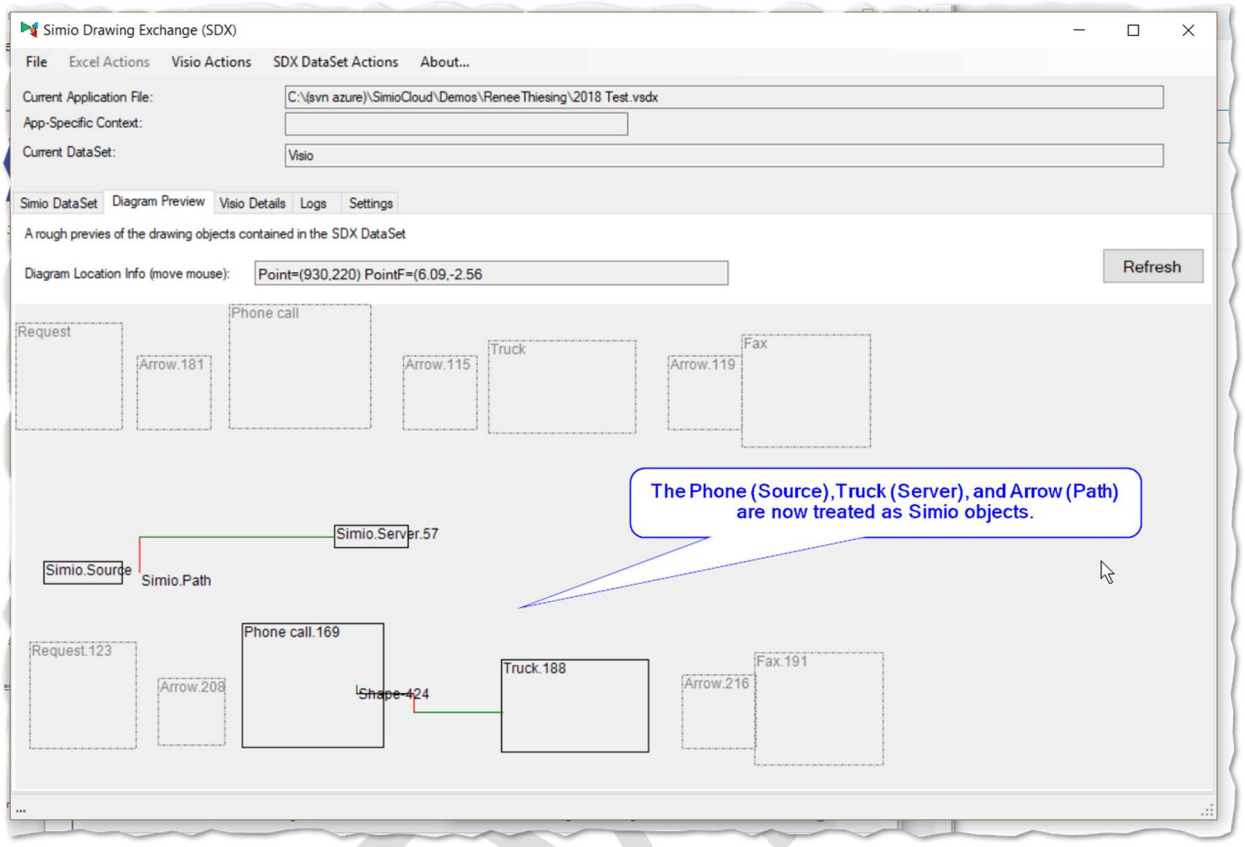


Figure 7- Shape-Data added to Phone and Truck. Note that they now have black outlines.

Simio
Forward Thinking

Use Cases

This section discusses some of the Use Cases considered for this Interface

Facility Import: Interface Visio Diagrams with Simio Facility View

This is perhaps the most asked-for interface and perhaps the most obvious one. Because of this, it is the first interface developed. Consider that you wish to draw a diagram using a standard Visio product to be later employed in a Simio model. This might be done during the design of a project where it is easier to demonstrate how the simulation would look to a project's stake-holders, but then periodically (or perhaps at the end of the project) quickly import it to Simio to implement the simulation.

Convert Visio Diagrams to Processes

In a similar way Processes could be constructed, so that instead of IntelligentObjects for the objects there would be Steps.

Convert Visio Diagrams to Tasks

In a similar way Tasks could be constructed, so that instead of IntelligentObjects for the objects there would be Tasks.

Future Road Map

There is nothing that prevents the reverse of these Use Cases. That is, exporting from Simio to a Visio diagram, but this is not planned in the first release.

Simio
Forward Thinking

Implementation

In this section we'll discuss the architecture in general, but specifically discuss the Visio to Simio Facility View Use Case as the example.

Exchanged Information

It is necessary when building any interface to determining the data that is needed, and how to extract that data.

The information that this Simio Add-In employs was derived from two existing sources:

1. The object, links, and vertex format already employed in the existing Simio Excel spreadsheets import, such as seen the Add-In ImportObjectsAndLinksFromSpreadsheet, and
2. Importing Objects and Links (Simio's Table-Base Objects, i.e. Auto-Create)

For example, for importing data into the Simio Facility model, we need to know (at a minimum) the following about the objects.

1. The Name
2. The Simio Class (the type of object, such as Source, Server, Sink, …)
3. The location in X,Y,Z space
4. The size (Width, Height, Depth/Length)
5. And any other property that we don't want to default.

And for the connections, the information needed is:

1. The Name
2. What type of connection, if any (e.g. Path, TimePath, …)
3. The beginning and end points on the Object
4. A list of segments (which may be only one segment).

For Visio, most data can be derived from standard Visio data. This includes name, location, size, connectors, etc. Thus, the essential data missing is the Simio names for the object classes, such as ObjectClass and LinkClass.

To get ObjectClass it is necessary to employ other techniques. Visio provides a convenient way to do this, which is essentially user-defined "Properties" of a Shape. In the Visio interface this seen as a feature that is (somewhat oddly) named "Shape-Data".

Note: Do not confuse Visio Shape-Data with the Visio Shape-Sheet. Internally, Shape-Data is the "user defined" part of a Shape-Sheet, and is what Simio uses to allow the user to associate custom data (e.g. Simio Object Classes) with a Shape. Think of Shape-Data as User-Defined Properties. There is more information about this in the Appendix.

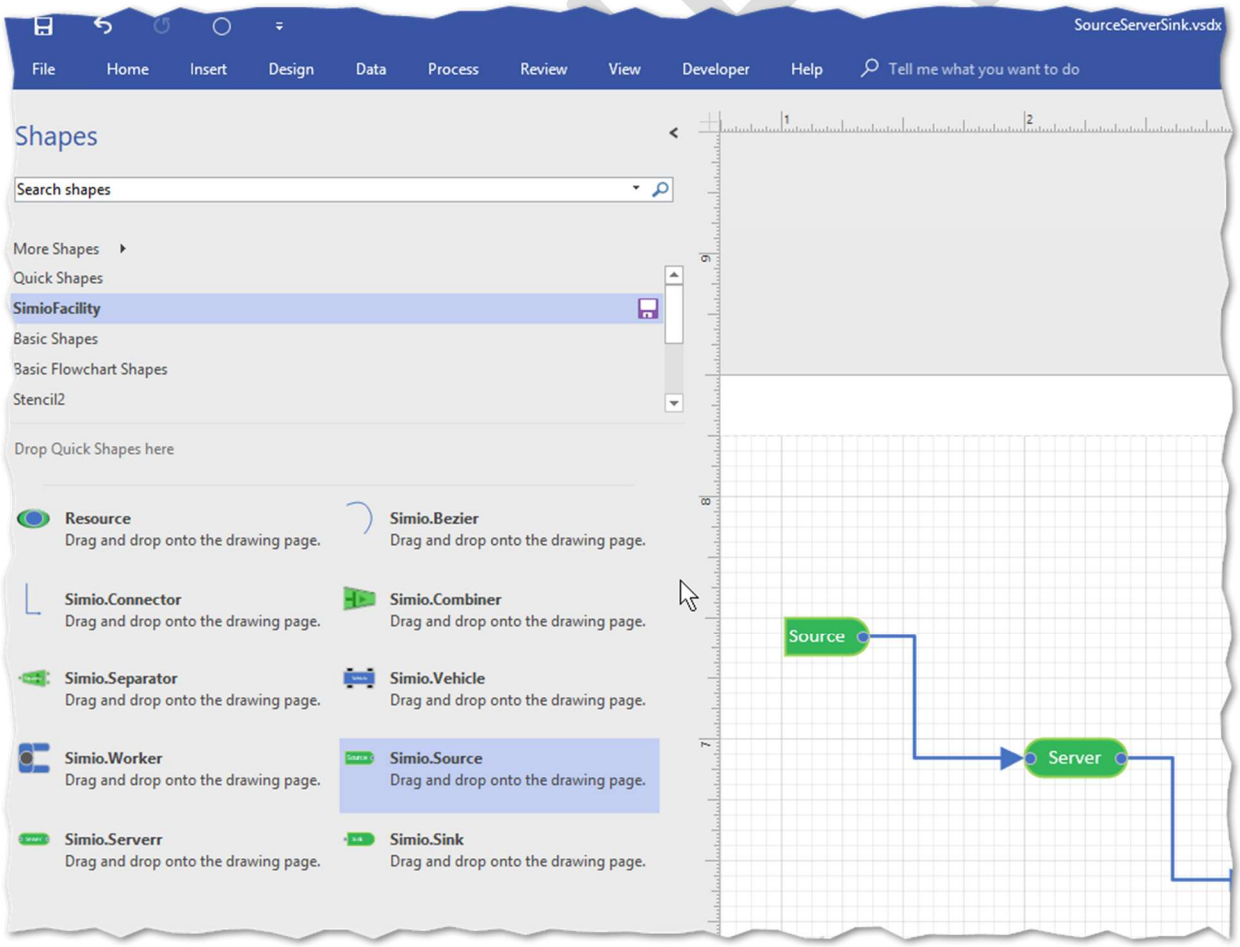
The conversion library needs to know the following information about an ObjectClass object:

- The Simio Object Name (e.g. “Server”)
- How many transfer nodes, and of what type (Input or Output)

For example, a Simio Server has two transfer nodes (1 input, 1 Output). A Source has one (Input) and a Separator has three (1 Input, 2 Output). This node information is used when validating the connecting LinkClass objects.

Our interface employs Shape-Data to store the Simio data (properties) that are needed, and by convention the ‘properties’ (Rows) of the Shape-Data used by Simio are prefixed with “Simio”. For example, the ObjectClass property that Simio needs can be found in Visio Shape-Data as the Name Simio_ObjectClass, and its corresponding “property” value (for example “Sink”.

The Simio Visio Stencils can be used to make this more convenient. The following diagram shows part of the Stencil that is used for creating Facility diagrams.



Simio provides custom Simio Stencils for Visio. These stencils already have these properties defined to make usage more convenient, but a user in a legacy situation with an already existing drawing need only add this Shape-Data to make the interface work.

This is done by doing a MRC (Mouse-Right-Click) on an object, and then following the steps as shown in this diagram:

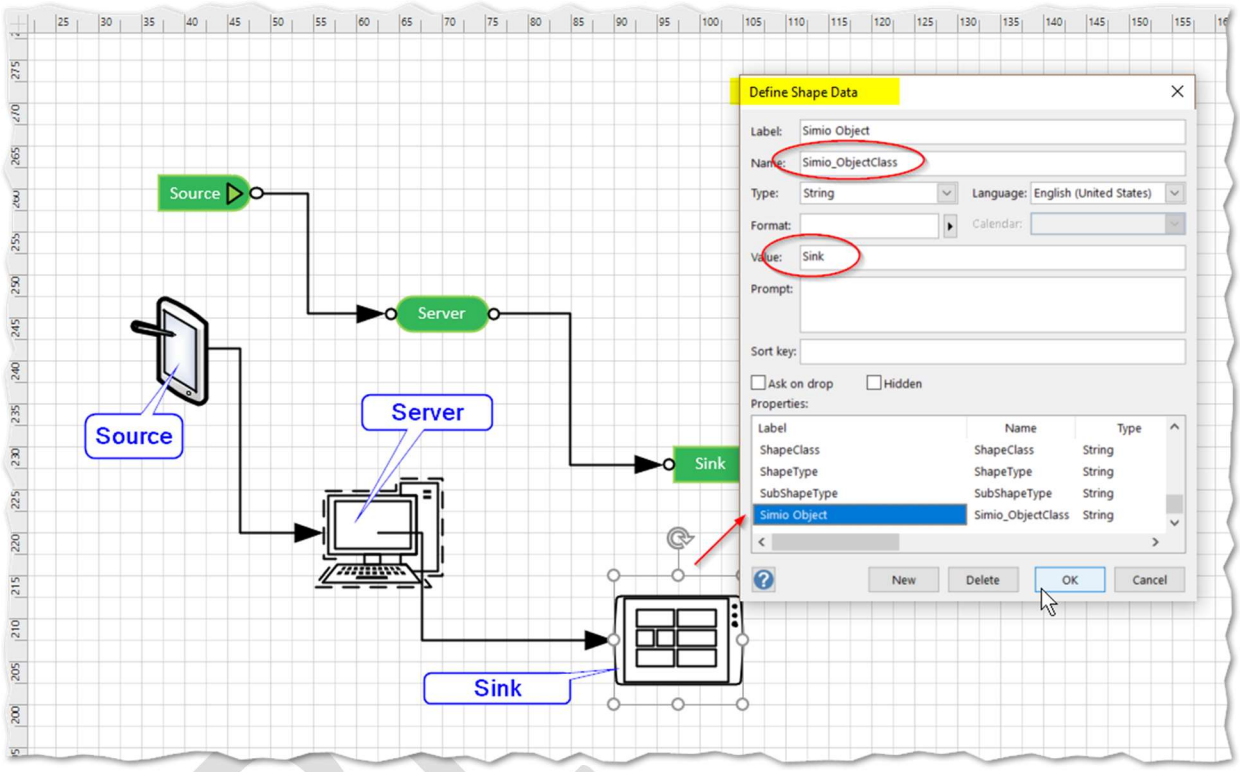


Figure 8 - Defining Properties (Shape-Data) for a Legacy Drawing

And then define the Simio Shape Data: (Note: The Name must begin with “Simio_”), and the Value must be one of the predefined values (Source, Server, Sink, …)

The result is the same as the Source-Server-Sink that was built with the Simio Stencil:

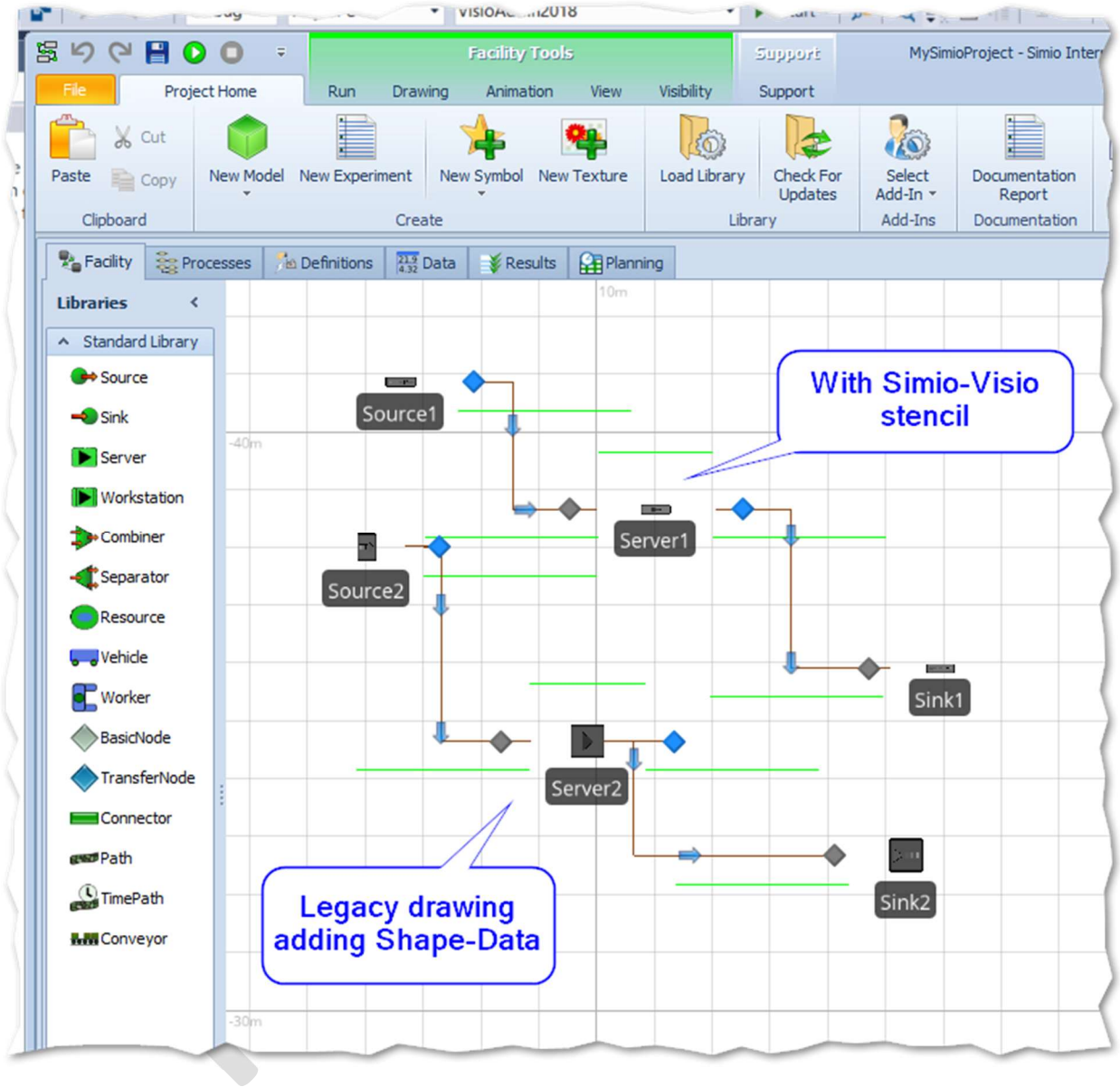
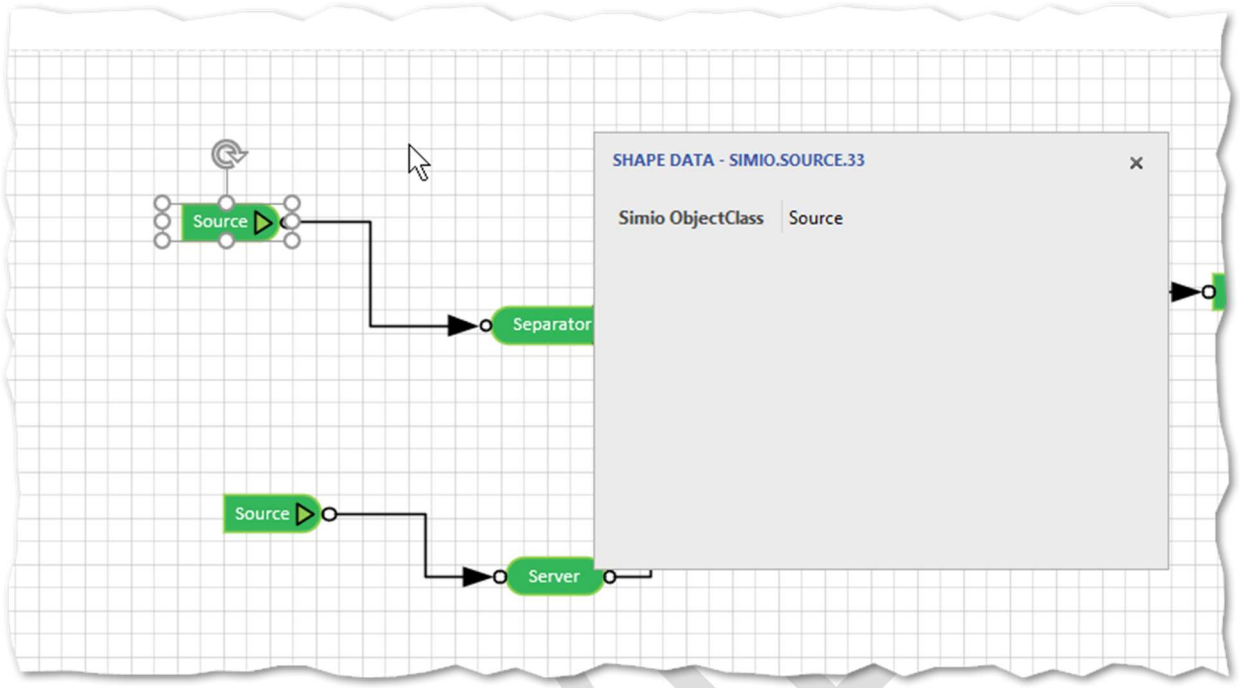


Figure 9 - Imported Facility using Legacy Visio Drawing

When this is done properly, the Shape will show Simo_ObjectClass with the value of “Source”:



Under the Covers – Legacy

Let’s look at our ‘legacy’ example of the Laptop as the Server. Within the package file under the Visio > Pages folder we’ll find a Page3.xml. Scanning through it we find our Laptop Shape:

Since we did not use a Simio Master stencil, all the information we need is here is in our Shape-Data, which is found under the XML Element ‘Section’ with the ‘N’ (Name) property ‘Property’. In the Row named ‘Simio_ObjectClass’ we just have to look in the Cell with ‘N’ of ‘Value’, and then pick up its ‘V’ value to get the fact that this is our Server.

Note that all the Name (‘N’) and (‘V’) convention used by Visio can get confusing, (I believe the reason Visio did this was because Name and Value are used so often that by abbreviating them to a single letter they were able to keep the file size smaller).

At any rate, this is all ‘under the cover’, so the provided interface software will do all this. *As a user, you need only to add the Shape-Data with the Simio conventions to your Visio-Shape.*

Connectors

Connectors are a different challenge. In Visio, connectors are just another type of Shape, and they can have more flexible geometry than Simio permits. But we can define the Visio connector as a Simio Link object (such as Connector, Path, etc.) provided by the Simio Facility Stencil to generate straight-line segments that can be employed by Simio.

Let's take the Legacy example first:

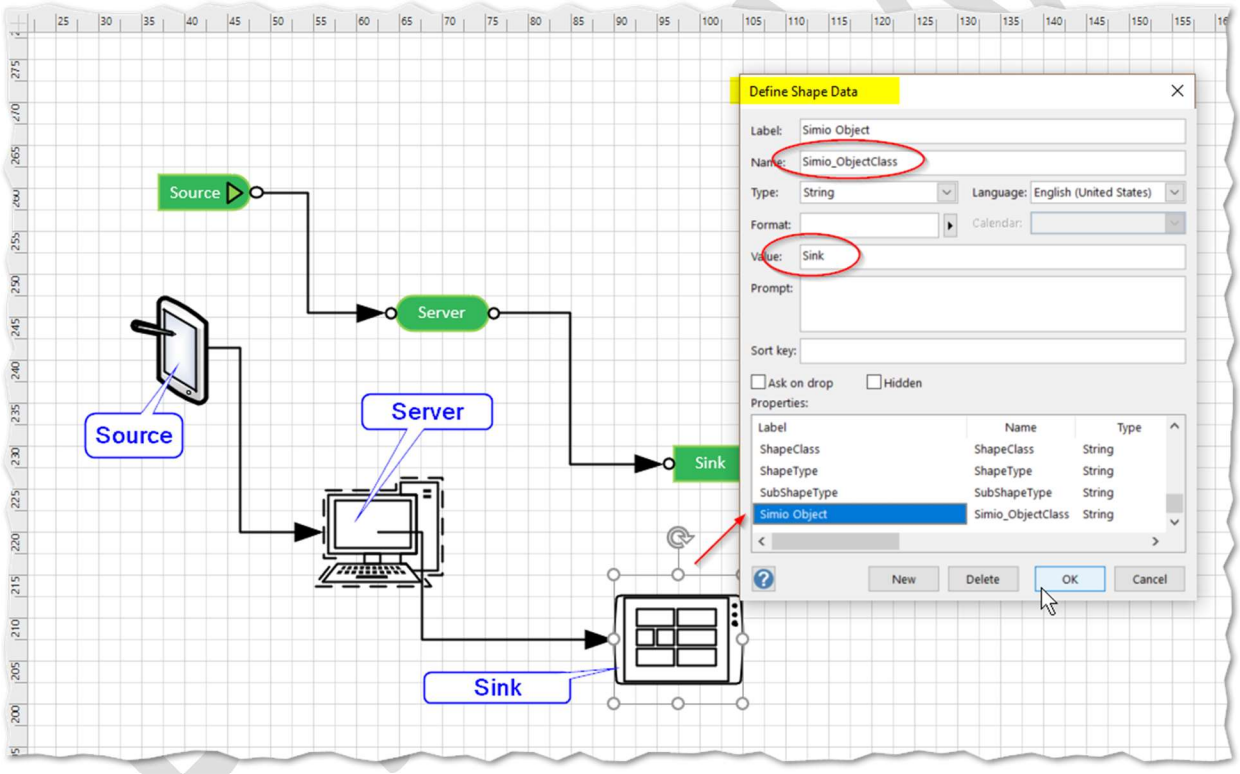


Figure 10 - Note: This Should be Simio_ObjectClass instead of Simio.Classname

The XML IDs are:

- Source (XML Web Service) ID=17
- Server (Laptop) ID=13
- Sink (Data) ID=31

- Source to Server Connector (ID=38)
- Server to Sink Connector (ID=37)

At the very end of the Page3.XML file is the ‘Connect’ Element:

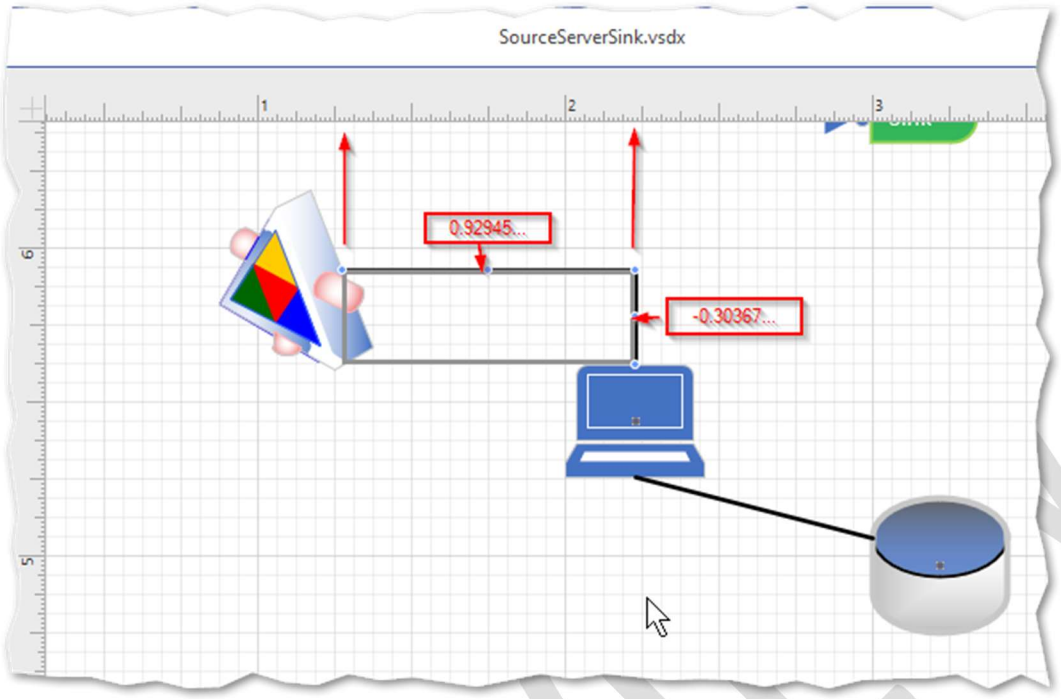
```
159     </Shapes>
160     <Connects>
161         <Connect FromSheet='38' FromCell='EndX' FromPart='12' ToSheet='13' ToCell='PinX' ToPart='3'/>
162         <Connect FromSheet='38' FromCell='BeginX' FromPart='9' ToSheet='17' ToCell='PinX' ToPart='3'/>
163         <Connect FromSheet='37' FromCell='EndX' FromPart='12' ToSheet='31' ToCell='PinX' ToPart='3'/>
164         <Connect FromSheet='37' FromCell='BeginX' FromPart='9' ToSheet='13' ToCell='PinX' ToPart='3'/>
165         <Connect FromSheet='12' FromCell='EndX' FromPart='12' ToSheet='44' ToCell='Connections.X3' ToPart='102'/>
166         <Connect FromSheet='12' FromCell='BeginX' FromPart='9' ToSheet='42' ToCell='Connections.X1' ToPart='100'/>
167         <Connect FromSheet='11' FromCell='EndX' FromPart='12' ToSheet='40' ToCell='Connections.X5' ToPart='104'/>
168         <Connect FromSheet='11' FromCell='BeginX' FromPart='9' ToSheet='48' ToCell='Connections.X1' ToPart='100'/>
169     </Connects>
170 </PageContents>
```

Here, we are interested only in the first four ‘Connect’ Elements.

Remembering that the Connectors are Shapes also, let’s look at ID=38, which connects the Source to the Server. The important items to note are the BeginX and EndX that are specified in the Connect Elements.

```
1073                </Row>
1074                <Row T='LineTo' IX='3' Del='1'/>
1075                <Row T='LineTo' IX='4' Del='1'/>
1076            </Section>
1077        </Shape>
1078        <Shape ID='38' NameU='Directory connector.38' Name='Directory connector.38' Type='Shape' Master='81'>
1079            <Cell N='PinX' V='1.751936740796923' F='Inh'/>
1080            <Cell N='PinY' V='5.773944822368422' F='Inh'/>
1081            <Cell N='Width' V='0.949251518406155' F='GUARD(EndX-BeginX)'/>
1082            <Cell N='Height' V='-0.3036738552631579' F='GUARD(EndY-BeginY)'/>
1083            <Cell N='LocPinX' V='0.4746257592030775' F='Inh'/>
1084            <Cell N='LocPinY' V='-0.151836927631579' F='Inh'/>
1085            <Cell N='BeginX' V='1.277310981593845' F='_WALKGLUE(BegTrigger,EndTrigger,WalkPreference)'/>
1086            <Cell N='BeginY' V='5.92578175' F='_WALKGLUE(BegTrigger,EndTrigger,WalkPreference)'/>
1087            <Cell N='EndX' V='2.2265625' F='_WALKGLUE(EndTrigger,BegTrigger,WalkPreference)'/>
1088            <Cell N='EndY' V='5.622107894736843' F='_WALKGLUE(EndTrigger,BegTrigger,WalkPreference)'/>
1089            <Cell N='BegTrigger' V='2' F='_XFTRIGGER(Sheet.17!EventXFMod)'/>
1090            <Cell N='EndTrigger' V='2' F='_XFTRIGGER(Sheet.13!EventXFMod)'/>
1091            <Cell N='TxtPinX' V='0.6264626868346563'/>
1092            <Cell N='TxtPinY' V='0'/>
1093            <Cell N='ConFixedCode' V='6'/>
1094            <Cell N='LineWeight' V='0.01388888888888889'/>
1095            <Section N='Property'>
1110            <Section N='Geometry' IX='0'>
1111                <Row T='MoveTo' IX='1'>
1112                    <Cell N='X' V='0'/>
1113                    <Cell N='Y' V='0'/>
1114                </Row>
1115                <Row T='LineTo' IX='2'>
1116                    <Cell N='X' V='0.949251518406155'/>
1117                    <Cell N='Y' V='0'/>
1118                </Row>
1119                <Row T='LineTo' IX='3'>
1120                    <Cell N='X' V='0.949251518406155'/>
1121                    <Cell N='Y' V='-0.3036738552631579'/>
1122                </Row>
1123                <Row T='LineTo' IX='4' Del='1'/>
1124            </Section>
1125        </Shape>
1126        <Shape ID='39' NameU='Simio.Server.84' Name='Simio.Server.84' Type='Group' Master='84'>
```

You can see in the diagram below that these match to the points where the connector Is glued to the Source and Sever.



And in the geometry, you can see that the movements are in relative coordinates, starting at 0,0 and them moving horizontally (X) about 0.949 units to the right and them moving down about -.303 units.

Simio
Forward Thinking

Appendix - Open Packaging Convention (OPC)

The Open Package Convention defines a standard way to store many types of data in a single container.

According to Wikipedia:

> *The Open Packaging Conventions is a container-file technology initially created by Microsoft to store a combination of XML and non-XML files that together form a single entity such as an Open XML Paper Specification document. OPC-based file formats combine the advantages of leaving the independent file entities embedded in the document intact and resulting in much smaller files compared to normal use of XML.*

Microsoft Office uses this convention for several product, including Visio (from Visio2013 forward).

Microsoft provides OPC package operations via API found in System.IO.Packaging.

You access (Open) a package like this:

```
35          // Get the Visio file from the location.
36          FileInfo[] fileInfos = dirInfo.GetFiles(fileName);
37          if (fileInfos.Count() > 0)
38          {
39              FileInfo fileInfo = fileInfos[0];
40              string filePathName = fileInfo.FullName;
41              // Open the Visio file as a package with
42              // read/write file access.
43              visioPackage = Package.Open(
44                  filePathName,
45                  FileMode.Open,
46                  FileAccess.ReadWrite);
47          }
48          // Return the Visio file as a package.
49          return visioPackage;
50      }
```

And then get its Parts like this:

```
// Open the Visio file in a Package object.
using (Package visioPackage = OpenPackage("DrawingSample.vsdx", Environment.SpecialFolder.MyDocuments))
{
    PackagePartCollection packageParts = visioPackage.GetParts();
    foreach (PackagePart part in packageParts)
    {
        var p1 = part;
        XDocument pageXML = GetXMLFromPart(part);
        if (pageXML == null)
            continue;
```

You can get data from a 'Part' as shown here to get an XDocument

```
1 reference
private static XDocument GetXMLFromPart(PackagePart packagePart)
{
    try
    {
        XDocument partXml = null;
        // Open the packagePart as a stream and then
        // open the stream in an XDocument object.
        Stream partStream = packagePart.GetStream();
        partXml = XDocument.Load(partStream);
        return partXml;

    }
    catch (Exception ex)
    {
        return null;
    }
}
```

More examples can be found in the Appendices specific to Office products such as Excel and Visio.

Simio
Forward Thinking

Appendix – Simio Diagram Exchange (SDX)

The Simio Diagram Exchange (SDX) is a standard format that drawings from external packages can be converted to so that Simio can use them.

It consists of four tables:

1. Properties (meta-data that helps Simio interpret the other three tables)
2. Objects – The objects that Simio must convert. For example, Intelligent object for the Facility View.
3. Links – The connectors that connect the Objects. Each link references two objects. For example, Path, TimePath, etc. in the Facility View.
4. Vertices – The geometry of the Links. Each vertex references. Each vertex references a Link and are an ordered set.

The default view for Simio is looking 'down' from up on the Y axis. The view is isometric so that it looks like a regular X,Y diagram (except that it is X,Z). It is thus a "right-handed" coordinate system.

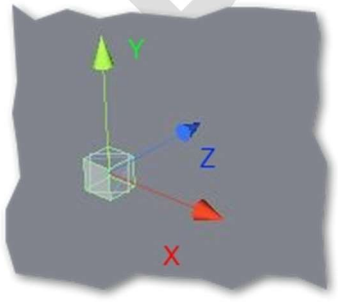
All coordinates use Simio conventions:

When looking 'down from above':

1. The origin is at the upper left
2. The horizontal axis is the X axis
3. The vertical axis is the Z axis
4. The Y axis is coming toward you (increments 'up' from the origin).

The nomenclature for 'Size' is:

- Width is along the X axis
- Length is along the Y axis ('Depth')
- Height is along the Z axis



Simio
Forward Thinking

Appendix – Visio Open Package Convention Document

This Appendix explains how Visio employs the Open Document Convention (OPC) file to implement its .VSDX files. The OPC structure is

1. Folder _rels
    a. File .rels
2. Folder docProps
    a. File app.xml
    b. File core.xml
    c. File custom.xml
3. Folder visio
    a. Folder _rels
    b. Folder masters – Contains masters.xml and master#.xml
    c. Folder media
    d. Folder pages – Contains pages.xml and page#.xml
    e. Folder themes – Contains files theme#.xml
    f. File document.xml
    g. File windows.xml – Contains Stencil references

Microsoft has extensive documentation about this format here:

https://docs.microsoft.com/en-us/office/client-developer/visio/introduction-to-the-visio-file-formatvsdx

Simio
Forward Thinking

Visio Components and Terminology

We are interested in extracting information from the Visio parts so that we can give enough information to the Simio API to build the Simio SDX DataSet (in XML format). Included in the SDX release is a sample builder (SDX Harness) that reads Visio diagrams and converts any selected Visio Page into a SDX DataSet.

This SDX DataSet can them be included in to your Simio project using the VisioAddIn.

Stencils and Pages

Stencils

Stencils contain Masters (or Master Shapes, if you prefer), and Masters are templates used to build Shapes on a Page.

Embedded in the file is data that is used to determine how Simio can use each Shape. Each Shape can have user-defined data (properties) associated with it. This information is stored in a Section called Shape-Data.

Pages

And here are some examples for getting items from the XDocument:

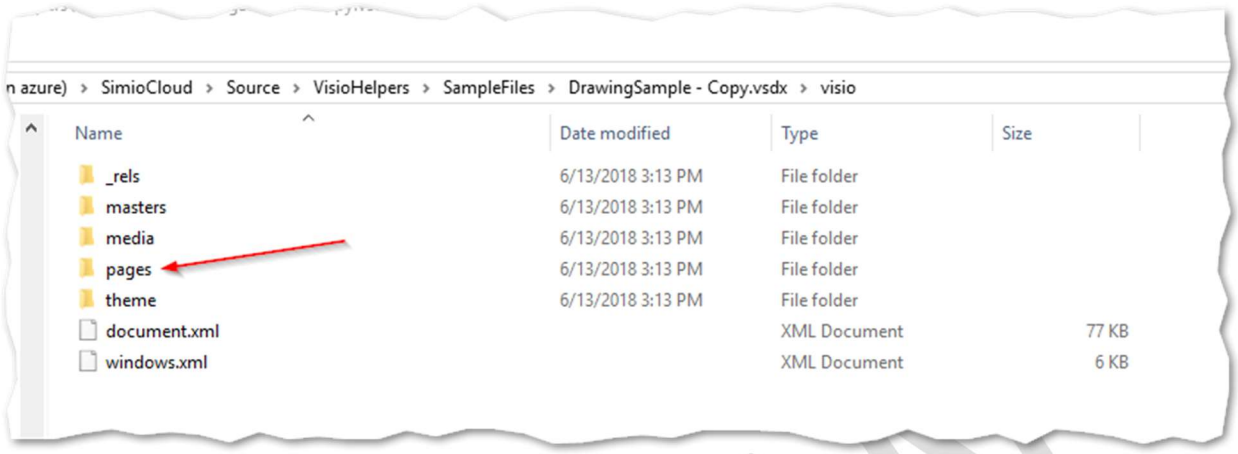
```
1 reference
private static IEnumerable<XElement> GetXElementsByName( XDocument packagePart, string elementType)
{
    // Construct a LINQ query that selects elements by their element type.
    IEnumerable<XElement> elements =
        from element in packagePart.Descendants()
        where element.Name.LocalName == elementType
        select element;
    // Return the selected elements to the calling code.
    return elements.DefaultIfEmpty(null);
}

1 reference
private static XElement GetXElementByAttribute(IEnumerable<XElement> elements,
        string attributeName, string attributeValue)
{
    if (!elements.Any())
        return null;

    // Construct a LINQ query that selects elements from a group
    // of elements by the value of a specific attribute.
    IEnumerable<XElement> selectedElements =
        from el in elements
        where el?.Attribute(attributeName)?.Value == attributeValue
        select el;
    // If there aren't any elements of the specified type
    // with the specified attribute value in the document,
    // return null to the calling code.
    return selectedElements?.DefaultIfEmpty(null).FirstOrDefault();
}
```

The structure of a Visio Project file (such as example.VSDX) can be viewed from File Explorer by changing the extension to ".zip", and then examining the file structure:

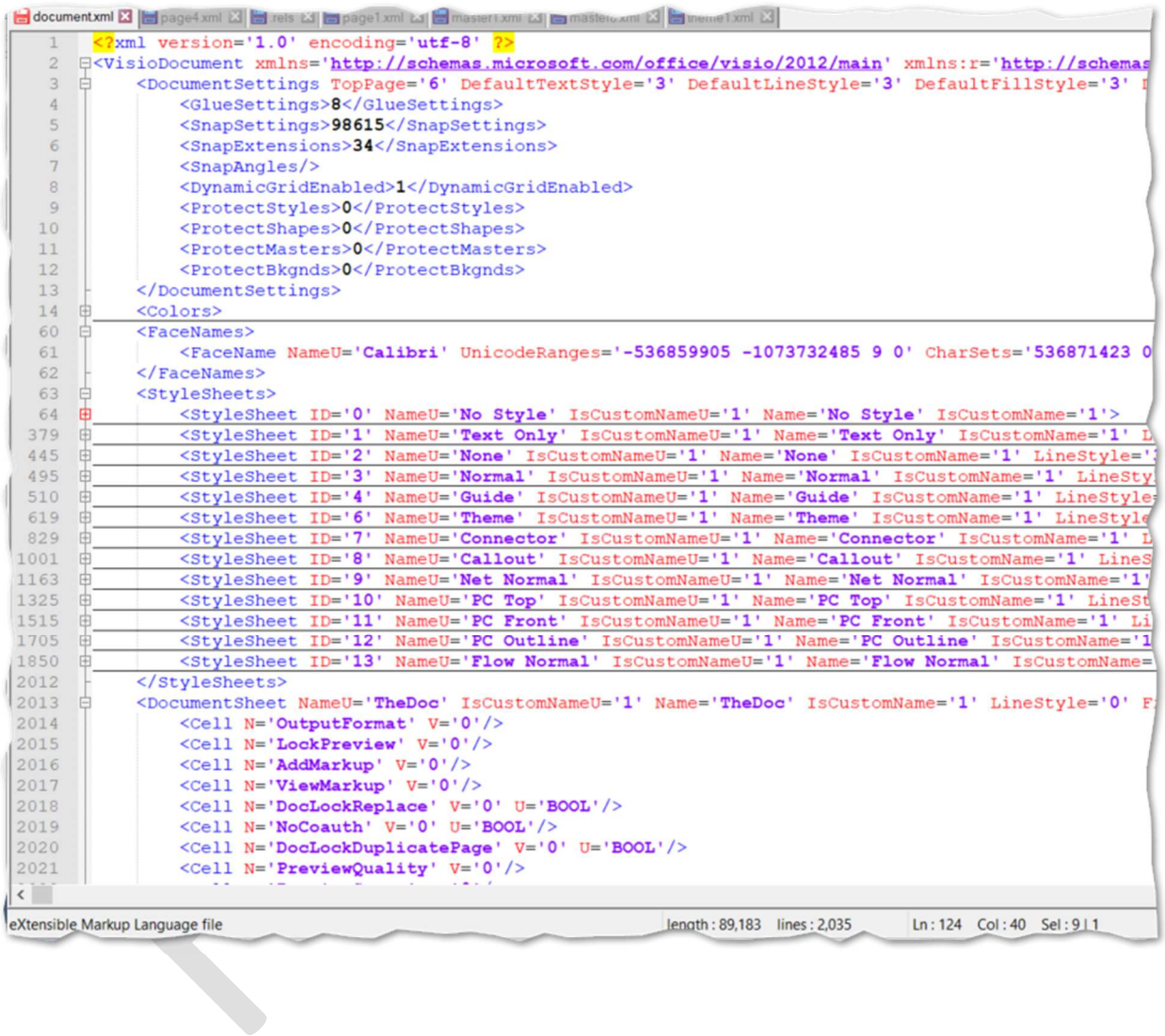
Simio
Forward Thinking



Simio
Forward Thinking

The Document.XML

The contained document.xml file describes the overall document settings, FaceNames (e.g. Calibri), StyleSheets, and the DocumentSheet.



Simio
Forward Thinking

The Windows.XML

This file describes the Visio Windows, which include child-windows such as Stencils. As seen, each 'Stencil' window references a ".vssx" file. These are also OPC files but differ from the ".vsdx" files in that they have no pages (no page{n}.xml files under the pages folder).

When you create your own stencils, they appear as ".vssx" extension files under "My Shapes", which is under your user's Document folder.

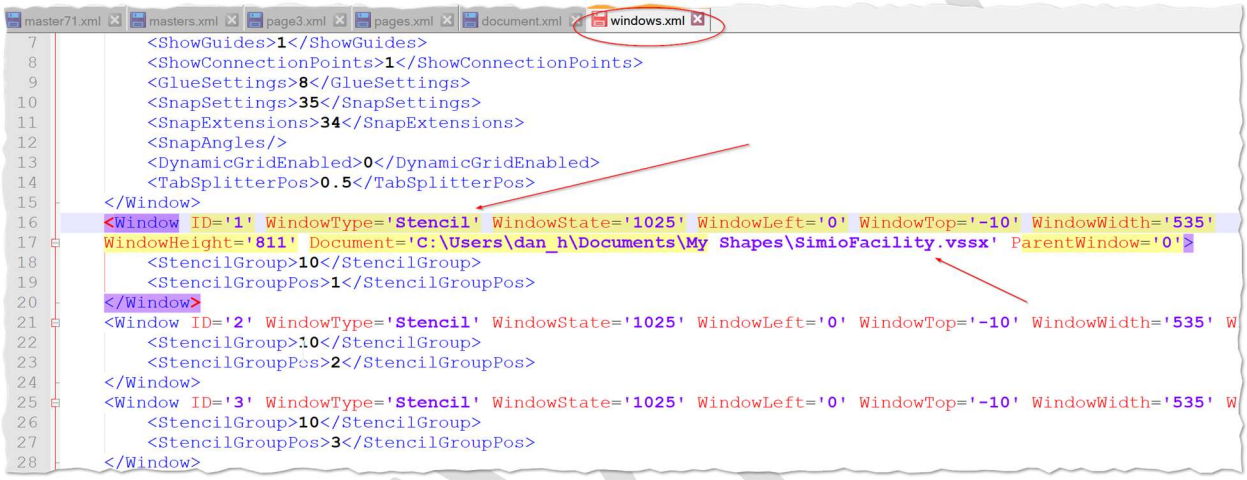
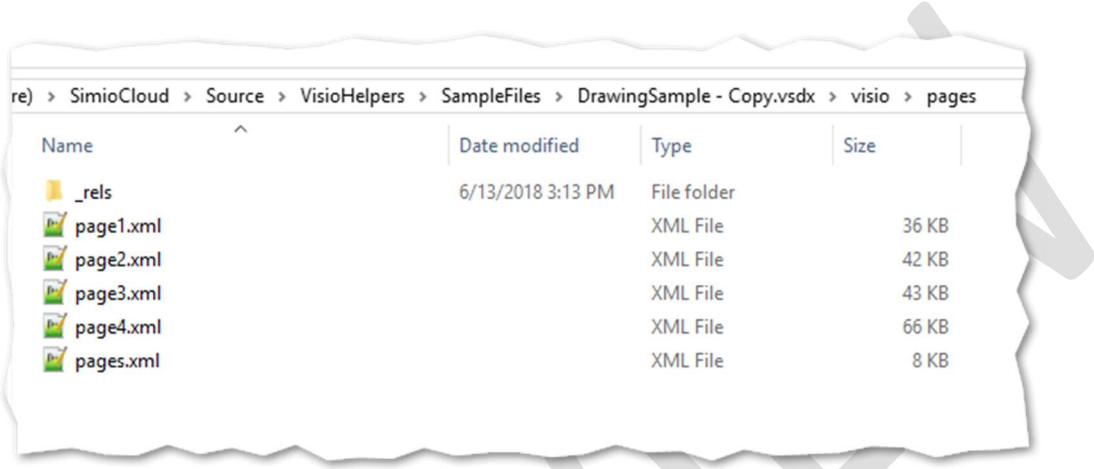


Figure 11 - Windows.xml File, Showing Stencils

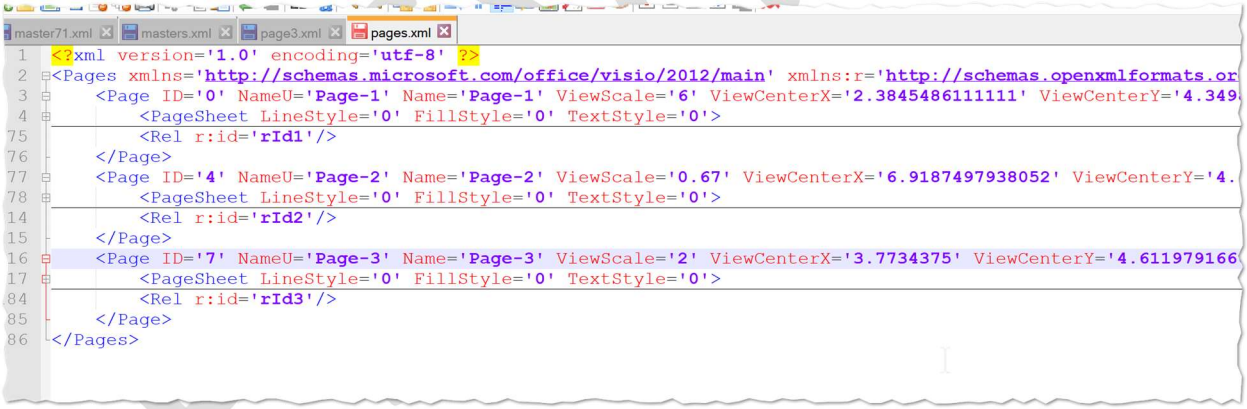
Simio
Forward Thinking

Pages Folder

Under the Pages folder are pages.xml and page{n}.xml files. Pages is a directory of all the page files. The individual page{n}.xml files in the pages folder is the motherload: it contains the drawings and correspond to the Page tables in the Visio application.



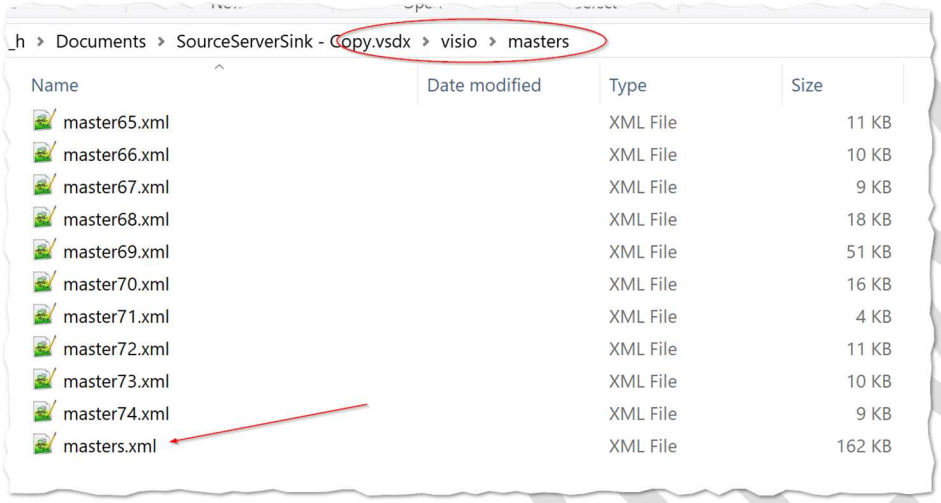
As mentioned, the pages.xml file contains a reference to all the individual 'page{n} files:



Simio
Forward Thinking

Masters Folder

The Masters folder has a structure like pages, with a file (master{n} for each Master shape (from the Stencil) and a global Masters.xml file:



Examining a Simple Page

It is instructive to examine a single simple page.

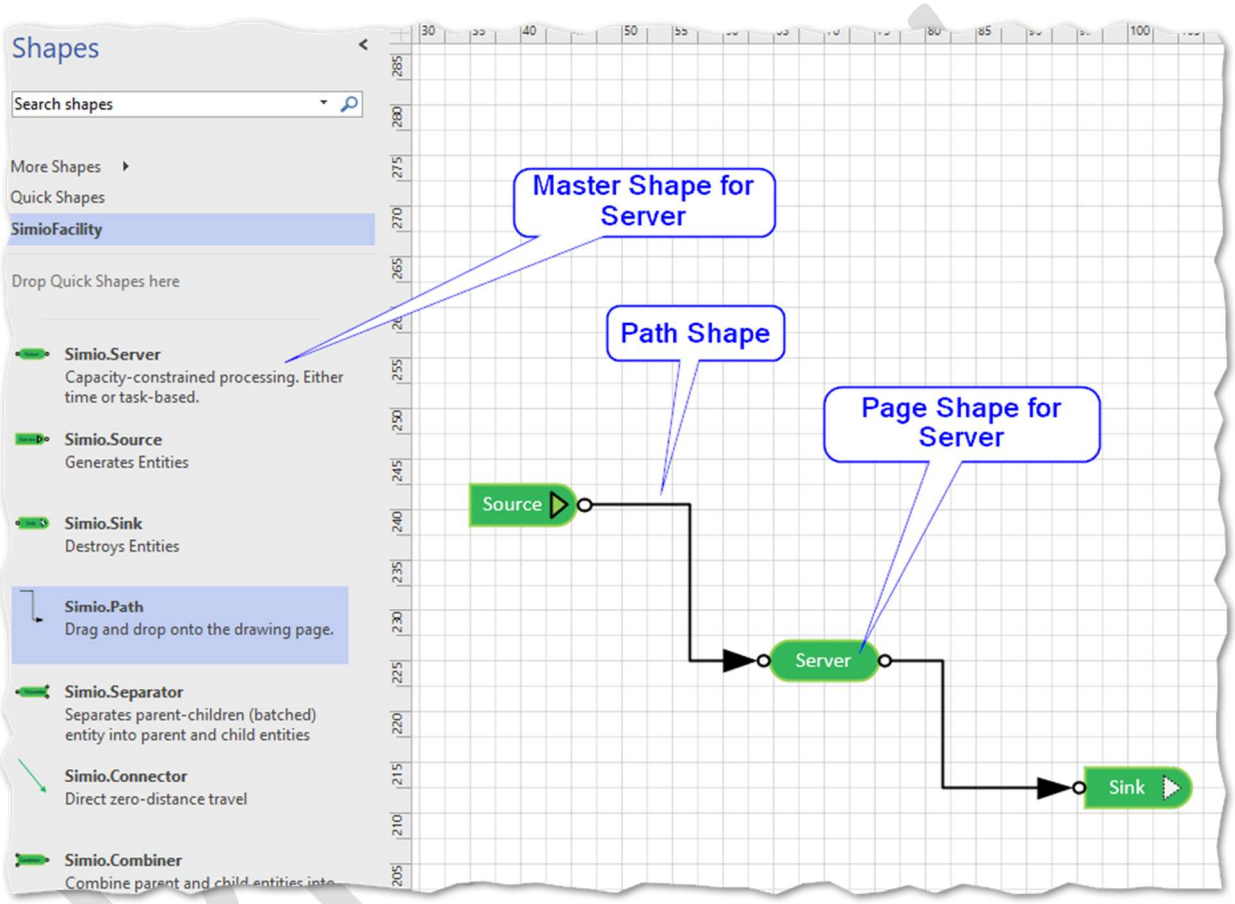


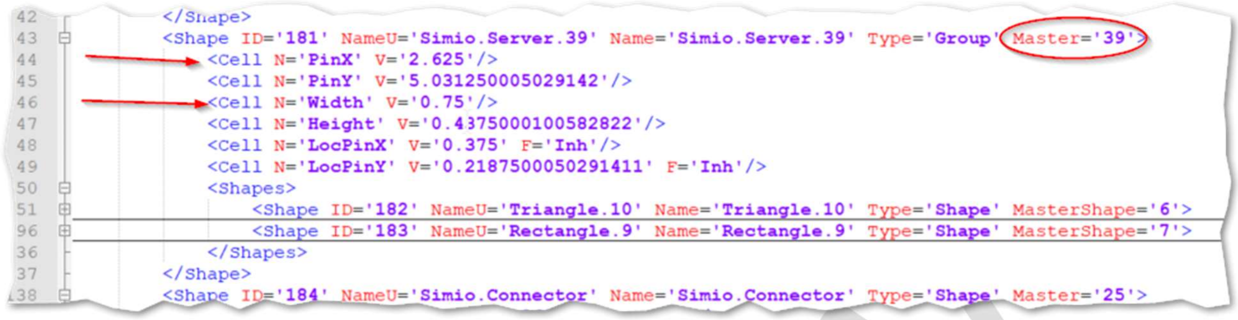
Figure 12 - A Simple Page

On the left is the Stencil (SimioFacility in this case). Each Shape in the Stencil is a Master. (as mentioned above, the Stencil itself – if custom – is found under User > Documents > MyShapes as a .VSSX file).

Each Shape in the diagram area is constructed from each “Master” shape from the Stencil. Each Master shape can be found by going to the “visio > masters” folder within the VSDX file and looking locating its master{n}.XML file (e.g. master23.xml)

The Shapes on the diagram Page are found within the folder “visio > pages” inside a page file (e.g. page1.XML).

Let’s look at the Server’s Shape first. Within page1.xml we find the following Shape section.



In this case the user had resized the Simio.Server Shape, so the Width and Height properties (Cells) exist for the Shape. But they would be missing if they had not, since Visio efficiently doesn’t bother since the default size is stored in the Master. In which case, we would have noted our Master ID (39) and opened the visio > master > masters.xml file and first located the Master with ID ‘39’.

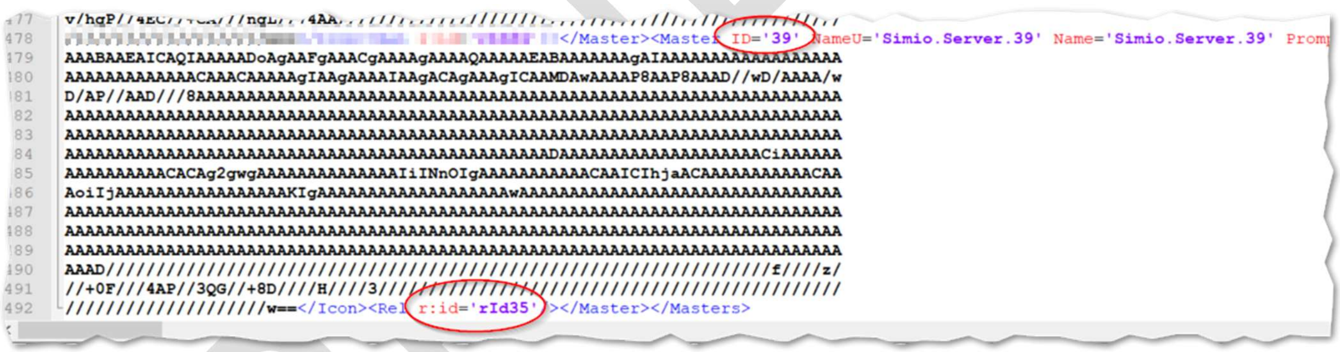
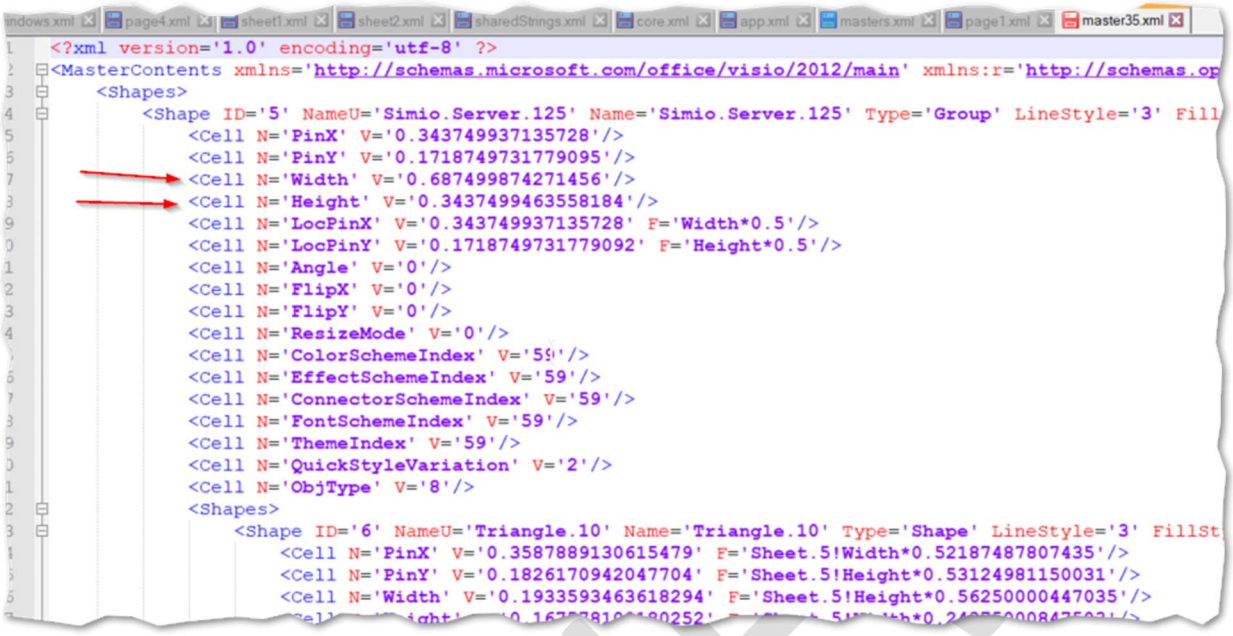


Figure 13 - Finding the Master file from the Shape

Note: the large encoded string is the icon used in the stencil.

Once we have located the Master, we can get the “Reference” ID of the Master the Reference ID (r:Id) which has the value rId35. And finally, we can parse this to get ‘35’ and then grab the Master35.xml file, to get the size dimensions (or any other Master default).



Again, only the contents of the Shape that differ from the Master are duplicated. And this is important because if – for example - we want to get the location and the size to pass them on to Simio, information like the location will always be in the Page Shape, but the size will only be in the Page file if the size was changed. If it hasn't changed, then we must go to the Master to fetch it.

To repeat, the procedure to find the Master is:

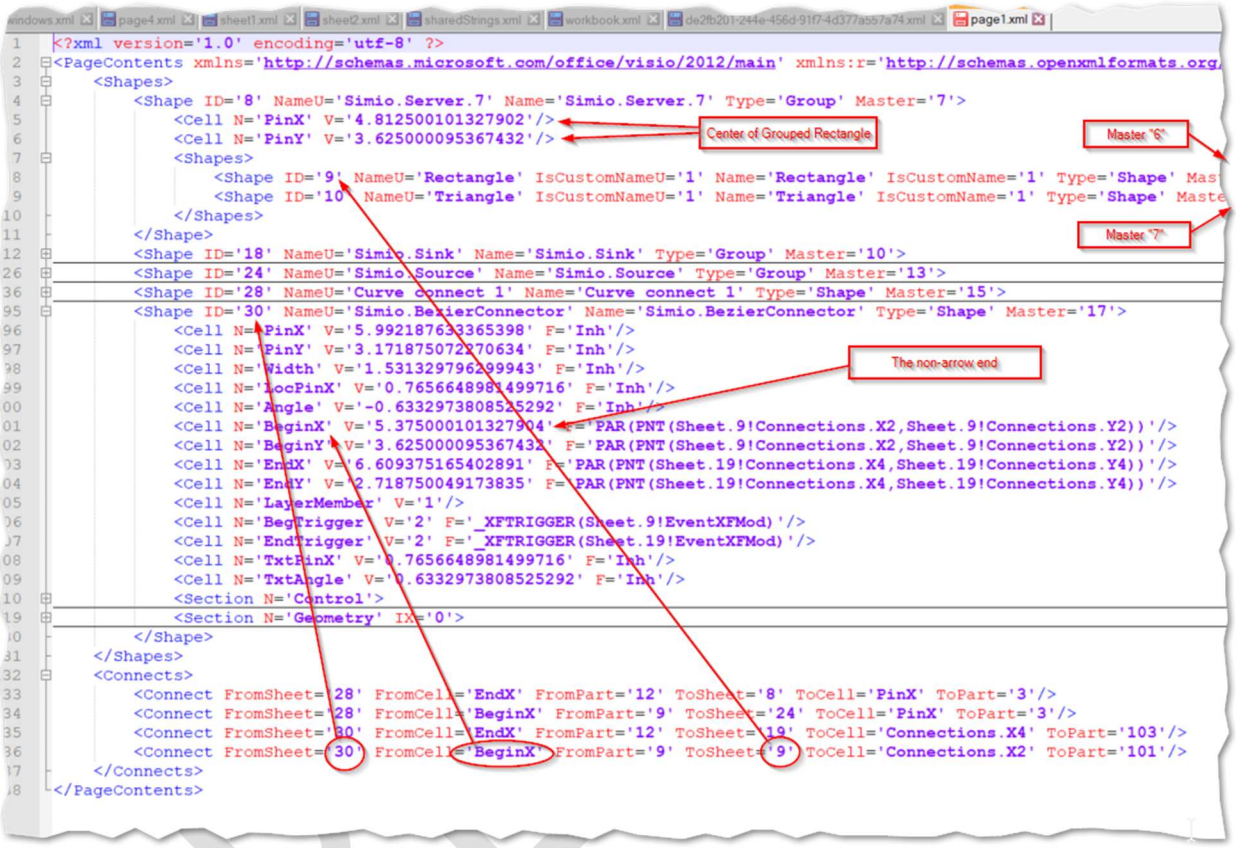
1. Find the Shape from the correct Pages folder.
2. Get the Master ID
3. Look in the Master.xml file in the Masters folder and find the r:id Attribute from the Rel Element (for example <Rel r:id="rId65" />
4. Get the id number after the 'rid' (for example 65)
5. Use this Rel id and find the Master{nn}.xml file (for example Master65.xml)
6. This is the Master that the Shape used.
7. If the Shape did not have a Property with a Label Simio.ObjectType, then the Master should have.

Connector Shapes

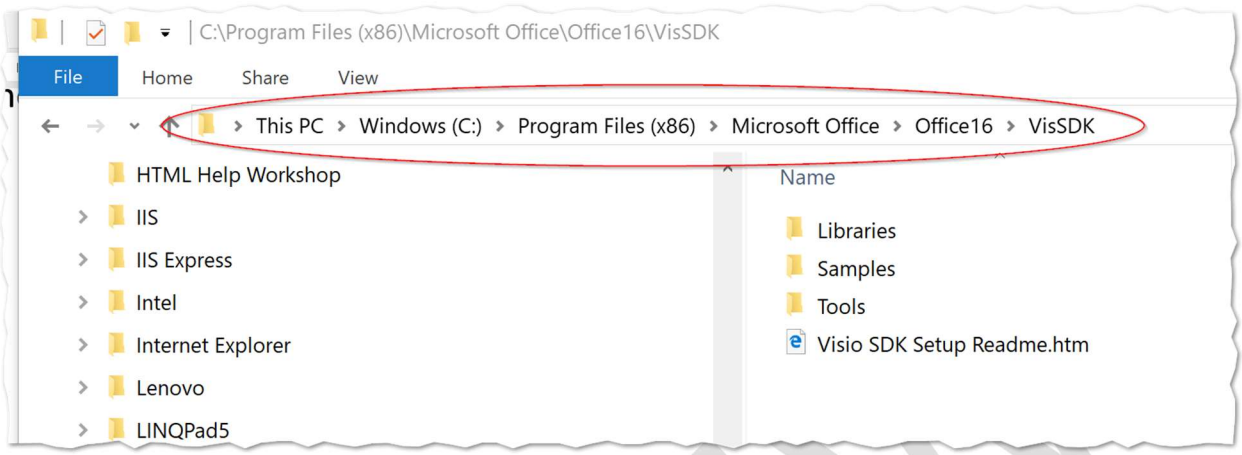
Let's now look at the Visio connectors, which become Simio Links. Connectors are also Visio Shapes, but have a special section at the end of each Page{n}.xml file with the element 'Connects'. Nested within Connects are a series of 'Connect' elements, each with a 'FromSheet' and 'ToSheet'.

Simio
Forward Thinking

Each connecting Shape has special Cell elements called BeginX, BeginY and EndX, EndY. Visio arbitrarily uses the BeginX and EndX in the Connect section, since to know the ‘X’ allows the selection of the ‘Y’ from the Shape section.



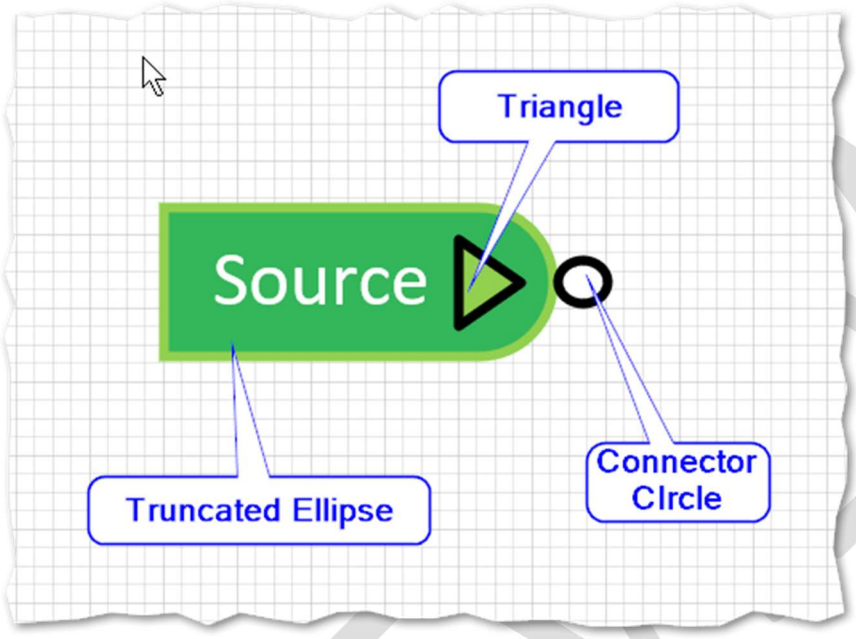
Simio
Forward Thinking



Simio
Forward Thinking

Example: The “Source” Shape

The Source is one of the simplest Shapes:



If we look at a “Source” Shape in the Page1.XML file, we can see that it has an ID of “1”, a Name of “Simio.Source.14” and reference to Master14, which is defined in the Masters.xml file:

Figure 14 - The 'Source' Shape within the Page1.xml file

This Master with an ID of 14 in Masters.xml then references the file Master6.xml (look at the Rel Element at the bottom) by specifying <Rel r:id='rId6' />:

Figure 15 - The 'Source' Master within Masters.xml

And Master6.xml contains all the template characteristics of this Master Shape. Remember that the Shape (with ID of '1') that referenced this (see above) had sub-Shape Elements with MasterShape attributes that referenced these shapes in this Master6.xml file.

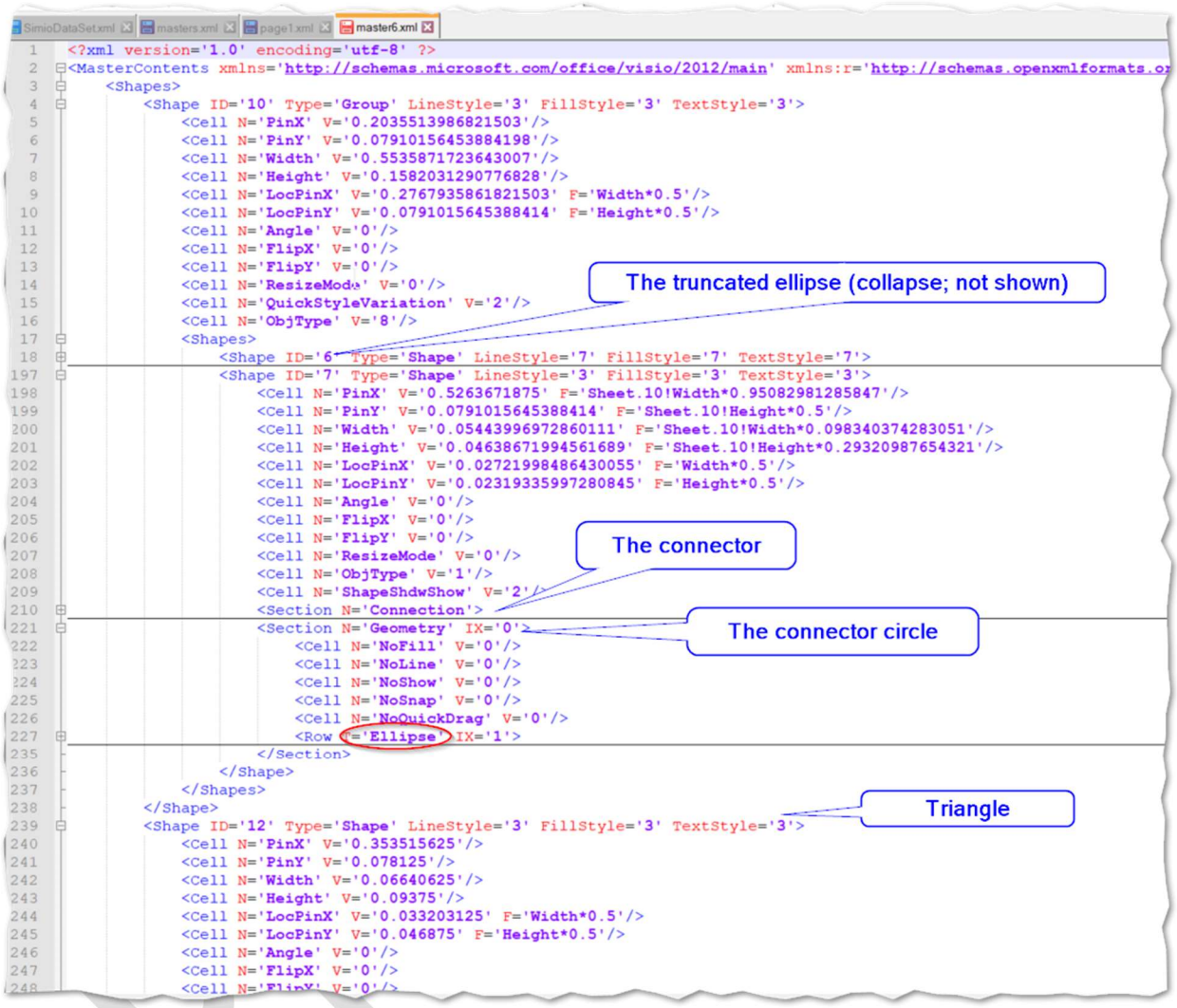


Figure 16 - The 'Source' Master (Master6.xml)

Simio
Forward Thinking

Appendix – Developer Resources

This appendix lists several resources available to the Developer, including some older interface options.

Videos from http://www.visguy.com

These videos provide an overview of Visio concepts. The YouTube locations are below.

- Visio, a flyby for developers - Module 1 - YouTube
- Visio, a flyby for developers - Module 2 - YouTube

Simio
Forward Thinking

Overview Outline

This outline just hits the tracks, modules and topic videos/lessons so you can quickly grasp the layout of the video series.

Track 1: Create Diagrams With Shapes and Text

Quick Start (5 Steps)
- Welcome to Visio 2016 (article)
- Create and Save (article)
- Format in Visio (article)
- Set Up Your Mobile Apps (article)
- Learn More (article)

Intro to Visio (5 videos)
- What is Visio (video)
- Using Visio Shapes (video)
- Change the Drawing Scale (video)
- Choose the Right View for the Task (video)
- Change Grid Spacing Size (video)

Create Diagrams (5 videos)
- Create a Diagram from a Template (video)
- Create a Flowchart (video)
- Create an Organizational Chart (video)
- Create a Floor Plan (video)
- Create a Network Diagram (video)

Add Shapes and Connectors (6 videos)
- Add and Format Shapes (video)
- Draw lines and Custom Shapes (video)
- Connect Shapes (video)
- Format Connectors (video)
- Measure a Distance in a Diagram (video)

  At time of publishing there was a bug here. The video link was incorrect, showing "Organize Shapes in Containers".
- Show Shape Size or Dimensions (video)

Add Text, Pictures and Tables (5 videos)
- Add Text to Shapes, Diagrams, or Connectors (video)
- Annotate a Diagram (video)
- Add Hyperlinks (video)
- Add Pictures (video)
- Create a Table (video)

These are online. For example: Custom Stencils: https://support.office.com/en-us/article/video-create-save-and-share-custom-stencils-b08cd9dd-58ee-48f7-889d-606d488c0e05?ui=en-US&rs=en-US&ad=US

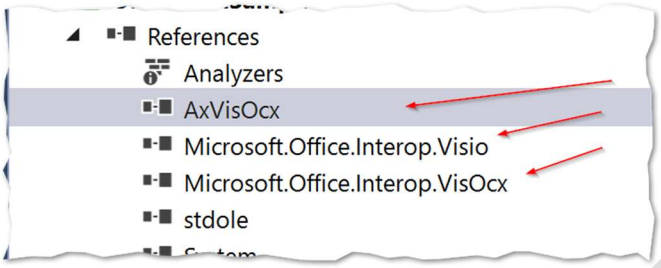
Simio
Forward Thinking

The Drawing Control

The drawing control is an older interface that depends upon OCX and Interop.

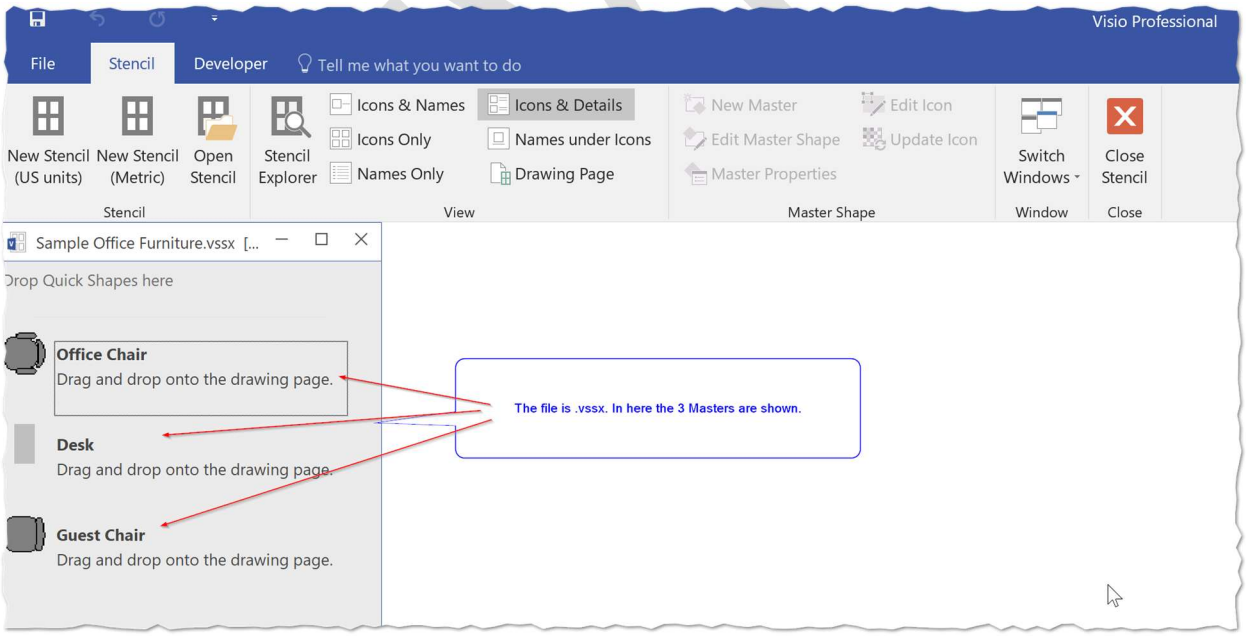
References:

There are 3 reference to the GAC:



Note that the VisOcx is the needed for the drawing control.

Here is code to find the masters in a visio diagram:



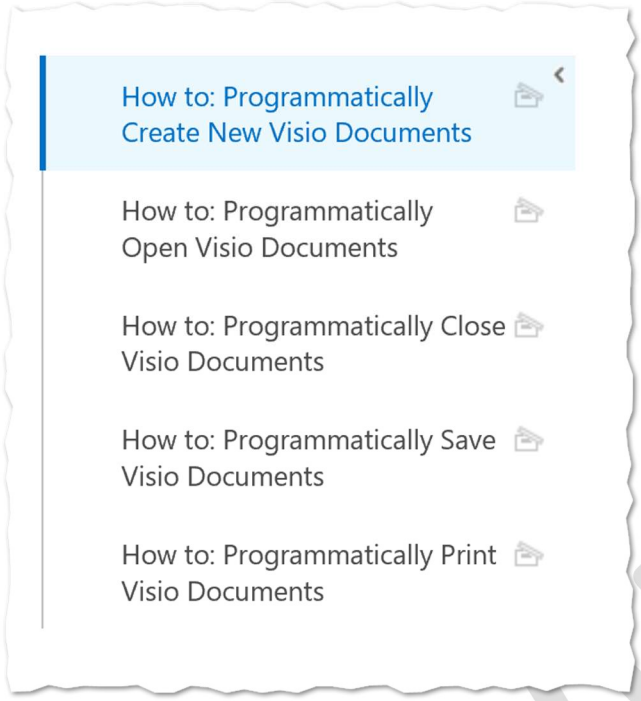
```
294     /// <summary>The GetMaster method gets the master by name.</summary>
295     /// <param name="drawingControl">Drawing control with the collection of
296     ///  Visio documents that contain the stencil and masters.</param>
297     /// <param name="stencilPath">The stencil path\filename.</param>
298     /// <param name="masterNameU">The universal name of the master.</param>
299     /// <returns>Master object if found. A COMException is thrown if not found.</returns>
300     [CLSCompliant(false)]
        1 reference
301     public static Master GetMaster(
302         AxMicrosoft.Office.Interop.VisOcx.AxDrawingControl drawingControl,
303         string stencilPath,
304         string masterNameU) {
305
306         // The drawing control object must be valid.
307         if (drawingControl == null) {
308
309             // Throw a meaningful error.
310             throw new ArgumentNullException("drawingControl");
311         }
312
313         Master targetMaster = null;
314
315         Document targetDocument;
316         Masters targetMasters;
317
318         targetDocument = OpenStencil(drawingControl, stencilPath);
319         targetMasters = targetDocument.Masters;
320         targetMaster = targetMasters.get_ItemU(masterNameU);  ≤ 3ms elapsed
321
322         return(targetMaster);
323     }
```

For example, the masterNameU might be "Office Chair", which would then return a reference to the Master of that name (using the get_ItemU() method.

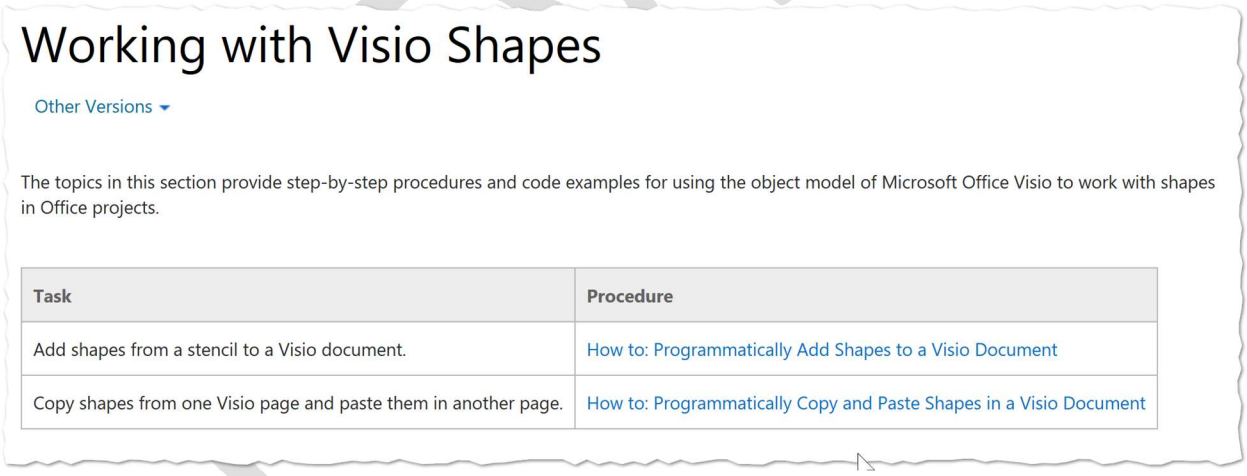
Visio Documents

https://msdn.microsoft.com/en-us/library/cc160751.aspx

Simio
Forward Thinking



https://msdn.microsoft.com/en-us/library/cc160747.aspx



Working with Visio Shapes

Other Versions

The topics in this section provide step-by-step procedures and code examples for using the object model of Microsoft Office Visio to work with shapes in Office projects.

Task	Procedure
Add shapes from a stencil to a Visio document.	How to: Programmatically Add Shapes to a Visio Document
Copy shapes from one Visio page and paste them in another page.	How to: Programmatically Copy and Paste Shapes in a Visio Document

Programmatically add Shapes

Simio
Forward Thinking

To add shapes to a Visio document

- With a document active, retrieve the masters from the Documents.Masters collection and drop the shapes on the active document. You can retrieve a master by using the index or master name.

  The following code example creates a blank Visio document, and then opens it with the **Basic Shapes** stencil docked. The code then retrieves several shapes and drops them on the active page.

C# | VB

```
this.Application.Documents.Add("");

Visio.Documents visioDocs = this.Application.Documents;
Visio.Document visioStencil = visioDocs.OpenEx("Basic Shapes.vss",
    (short)Microsoft.Office.Interop.Visio.VisOpenSaveArgs.visOpenDocked);

Visio.Page visioPage = this.Application.ActivePage;

Visio.Master visioRectMaster = visioStencil.Masters.get_ItemU(@"Rectangle");
Visio.Shape visioRectShape = visioPage.Drop(visioRectMaster, 4.25, 5.5);
visioRectShape.Text = @"Rectangle text.";

Visio.Master visioStarMaster = visioStencil.Masters.get_ItemU(@"Star 7");
Visio.Shape visioStarShape = visioPage.Drop(visioStarMaster, 2.0, 5.5);
visioStarShape.Text = @"Star text.";

Visio.Master visioHexagonMaster = visioStencil.Masters.get_ItemU(@"Hexagon");
Visio.Shape visioHexagonShape = visioPage.Drop(visioHexagonMaster, 7.0, 5.5);
visioHexagonShape.Text = @"Hexagon text.";
```

"All of the objects and members in the VBA object model reference correspond to types and members in the Visio primary interop assembly (PIA). For example, the **Document** object in the VBA object model reference corresponds to the **Microsoft.Office.Interop.Visio.Document** type in the Visio PIA. Although the VBA object model reference provides code examples for most properties, methods, and events, you must translate the VBA code in this reference to Visual Basic or Visual C# if you want to use them in a Visio VSTO Add-in project that you create by using Visual Studio,"

How a program uses automation to control Visio

A program controls Visio by accessing its objects and then using their properties, methods, and events.

- *Objects* represent items you work with in the Visio application, such as documents, drawing pages, shapes, and cells containing formulas.
- *Properties* are attributes that determine the appearance or behavior of objects. For example, a **Shape** object has a **Name** property, which represents the name of that shape.
- *Methods* are actions provided by an object. For instance, a program can perform the **Add** method on a **Page** object. This is the same as adding a page to a document by clicking **Blank Page** on the **Insert** tab.
- *Events* trigger code or entire programs. For example, an event can programmatically trigger code when a document is opened or trigger a program when a shape is double-clicked.

Simio
Forward Thinking

Appendix – Imported Data Conventions

Objects

- ObjectClass
- ObjectName
- X,Y,Z (doubles)
- Length, Width, Height (doubles)
- EntityType
- InitialNumberInSystem
- RideOnTransporter
- TransporterName
- PropertyA
- PropertyB
- PropertyC

Links

- LinkClass (e.g. Path)
- LinkName
- FromNode
- ToNode
- Network
- Width, Height
- Type (Bidirectional)
- PropertyA
- PropertyB
- PropertyC

Vertices

LinkName

VertexX

VertexY

VertexZ

Simio
Forward Thinking

Appendix – Excel OPC Format

Excel in OPC Format:

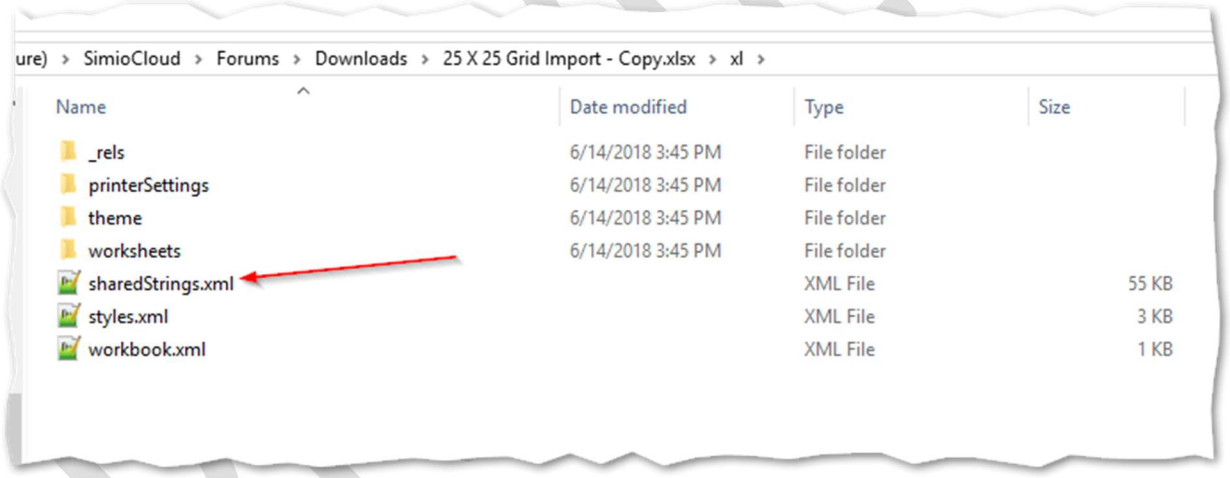
Excel (after 2007, at least) is all about representing a series of Worksheets, which are – in turn – a series of sparse matrices.

Like many Microsoft Office Products, Excel employs the Open Package Convention format for storing information.

For Simio, the Worksheets can be of the following types:

1. Object
2. Links
3. Vertices

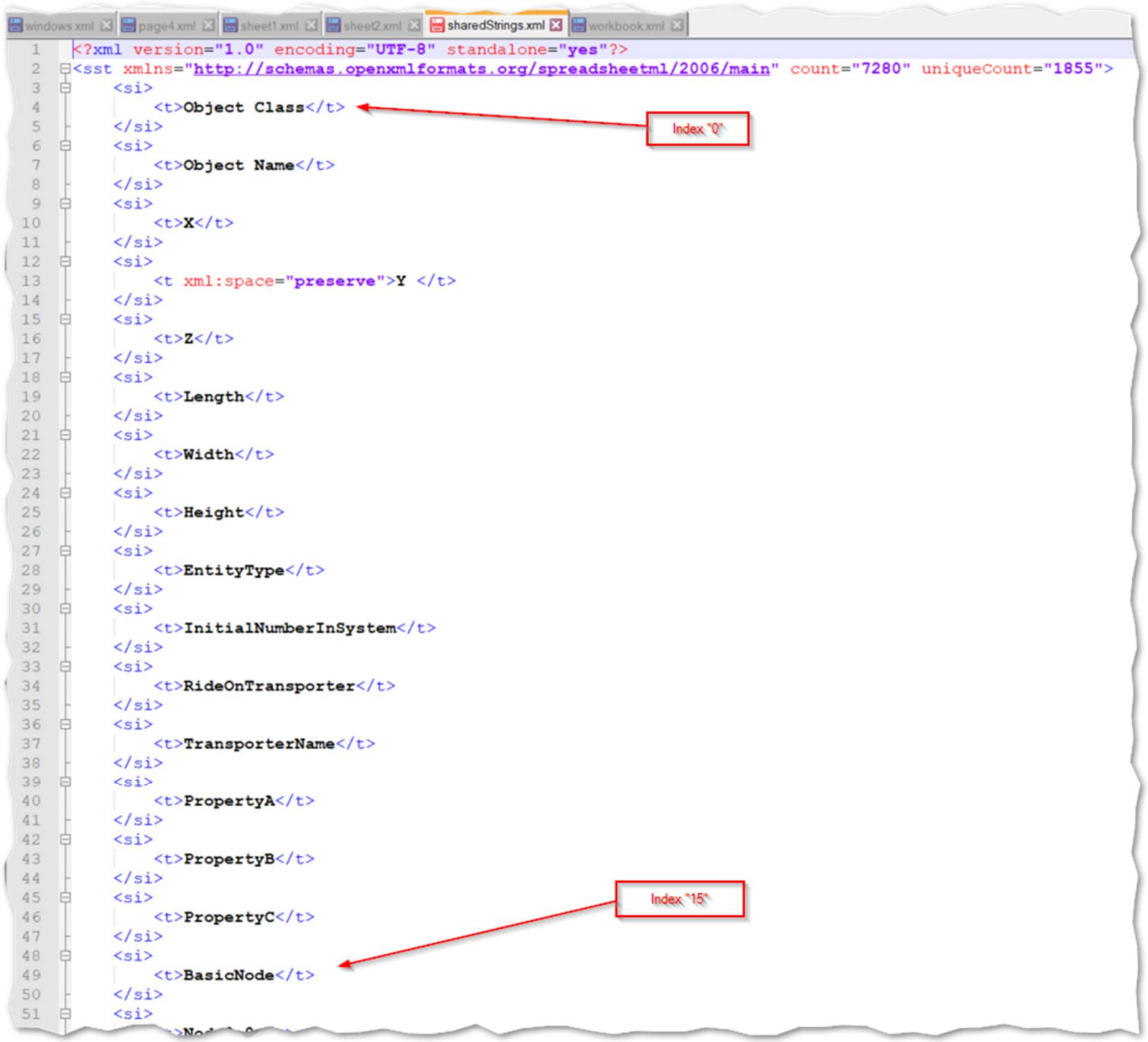
Under XL



Simio
Forward Thinking

XML File: sharedString

The odd duck here is the sharedStrings.XML, which contains a list of unique string. A guess as to why this is may be that in Excel many strings are often repeated, so the developers felt that this would increase storage efficiency.

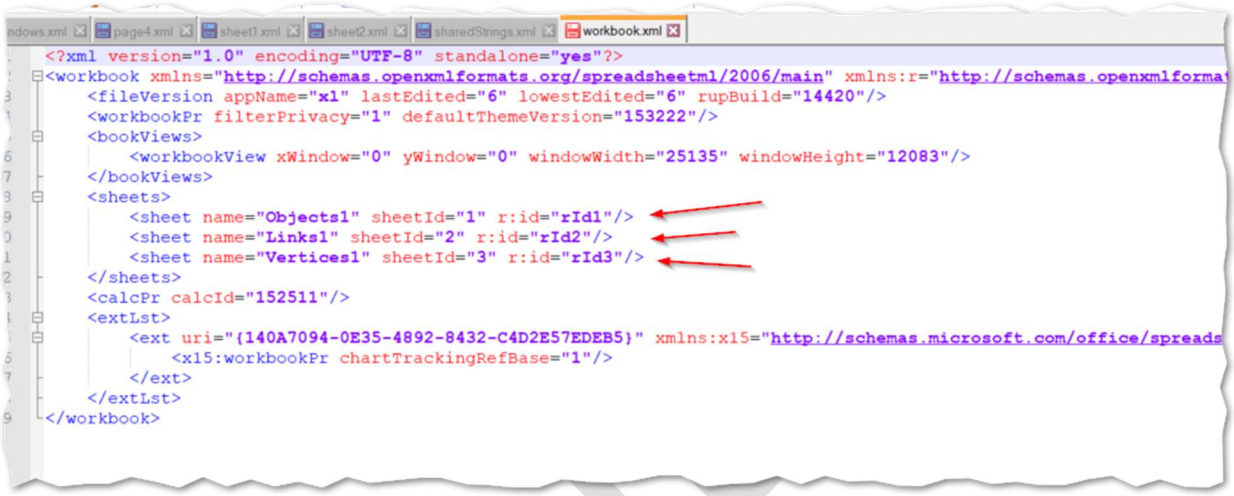



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="7280" uniqueCount="1855">
    <si>
        <t>Object Class</t>
    </si>
    <si>
        <t>Object Name</t>
    </si>
    <si>
        <t>X</t>
    </si>
    <si>
        <t xml:space="preserve">Y </t>
    </si>
    <si>
        <t>Z</t>
    </si>
    <si>
        <t>Length</t>
    </si>
    <si>
        <t>Width</t>
    </si>
    <si>
        <t>Height</t>
    </si>
    <si>
        <t>EntityType</t>
    </si>
    <si>
        <t>InitialNumberInSystem</t>
    </si>
    <si>
        <t>RideOnTransporter</t>
    </si>
    <si>
        <t>TransporterName</t>
    </si>
    <si>
        <t>PropertyA</t>
    </si>
    <si>
        <t>PropertyB</t>
    </si>
    <si>
        <t>PropertyC</t>
    </si>
    <si>
        <t>BasicNode</t>
    </si>
    <si>
```

Simio
Forward Thinking

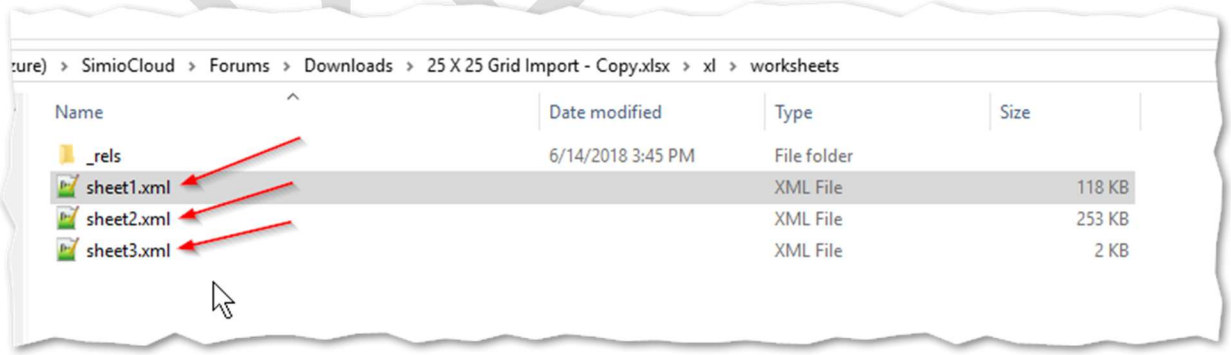
WorkBook File

Ties the WorkSheets together:



WorkSheets

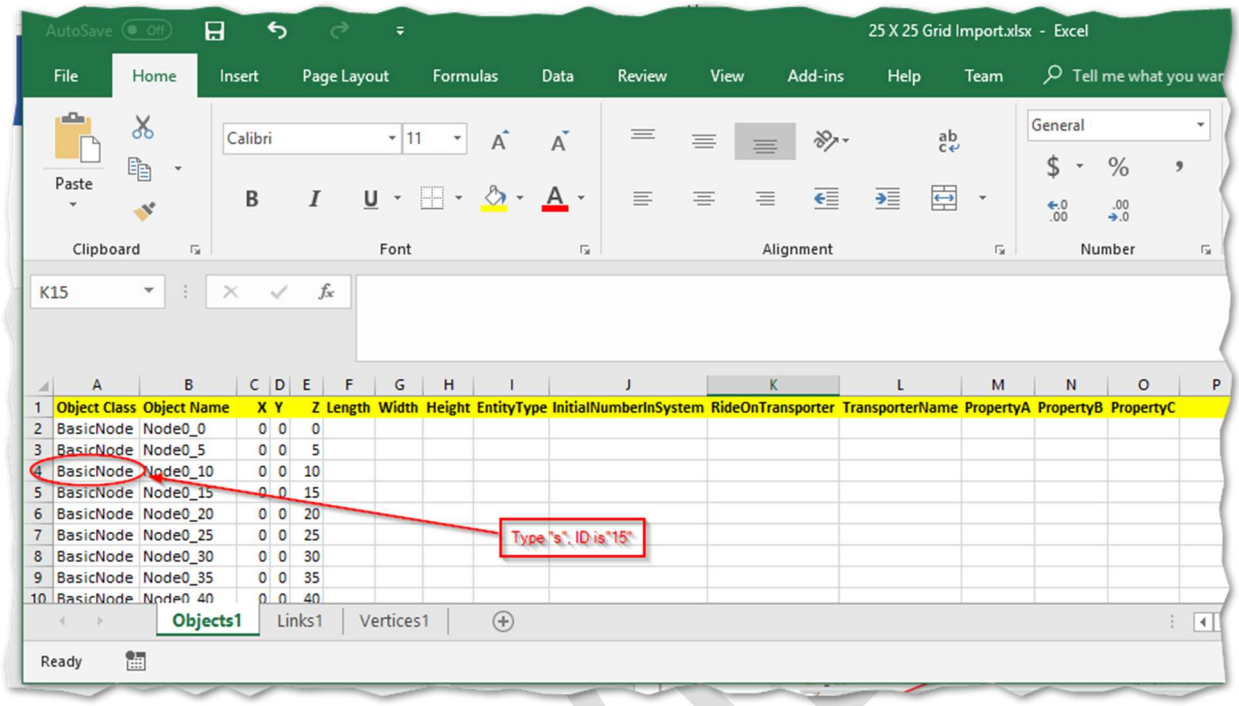
Under XL is the WorkSheets folder



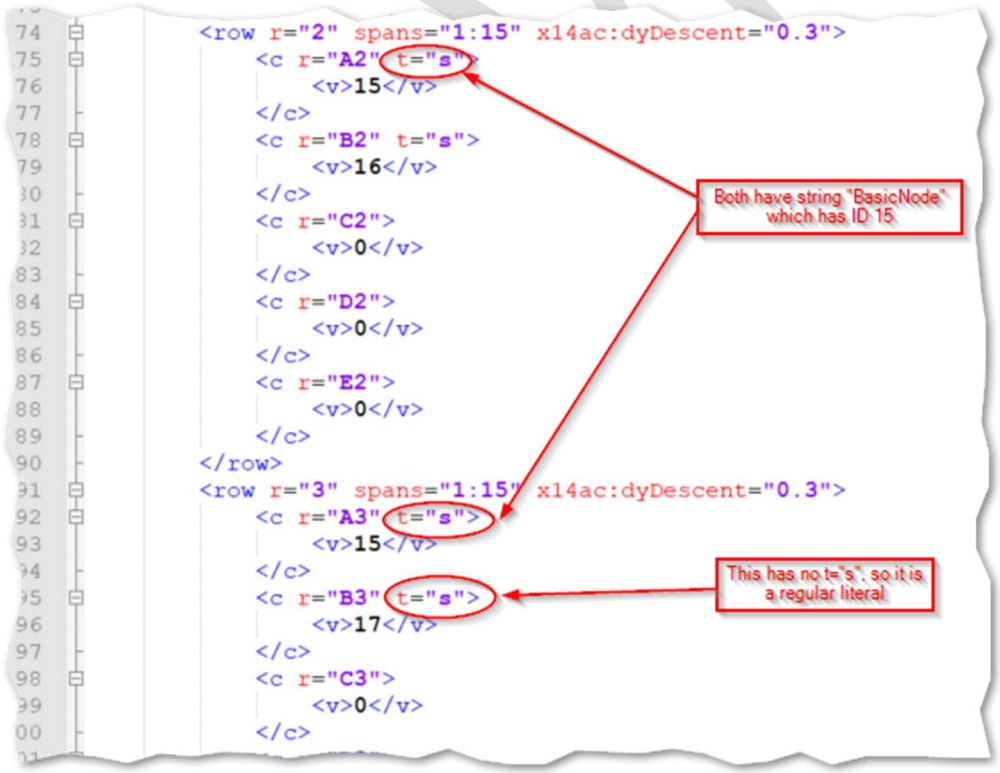
The Excel has three Worksheets: Objects, Links, and Vertices

Each type of worksheet has the Property names in the first row, and then “records” for the remaining.

The first column (field) must always contain a value.



Object Class	Object Name	X	Y	Z	Length	Width	Height	EntityType	InitialNumberInSystem	RideOnTransporter	TransporterName	PropertyA	PropertyB	PropertyC
BasicNode	Node0_0	0	0	0										
BasicNode	Node0_5	0	0	5										
BasicNode	Node0_10	0	0	10										
BasicNode	Node0_15	0	0	15										
BasicNode	Node0_20	0	0	20										
BasicNode	Node0_25	0	0	25										
BasicNode	Node0_30	0	0	30										
BasicNode	Node0_35	0	0	35										
BasicNode	Node0_40	0	0	40										

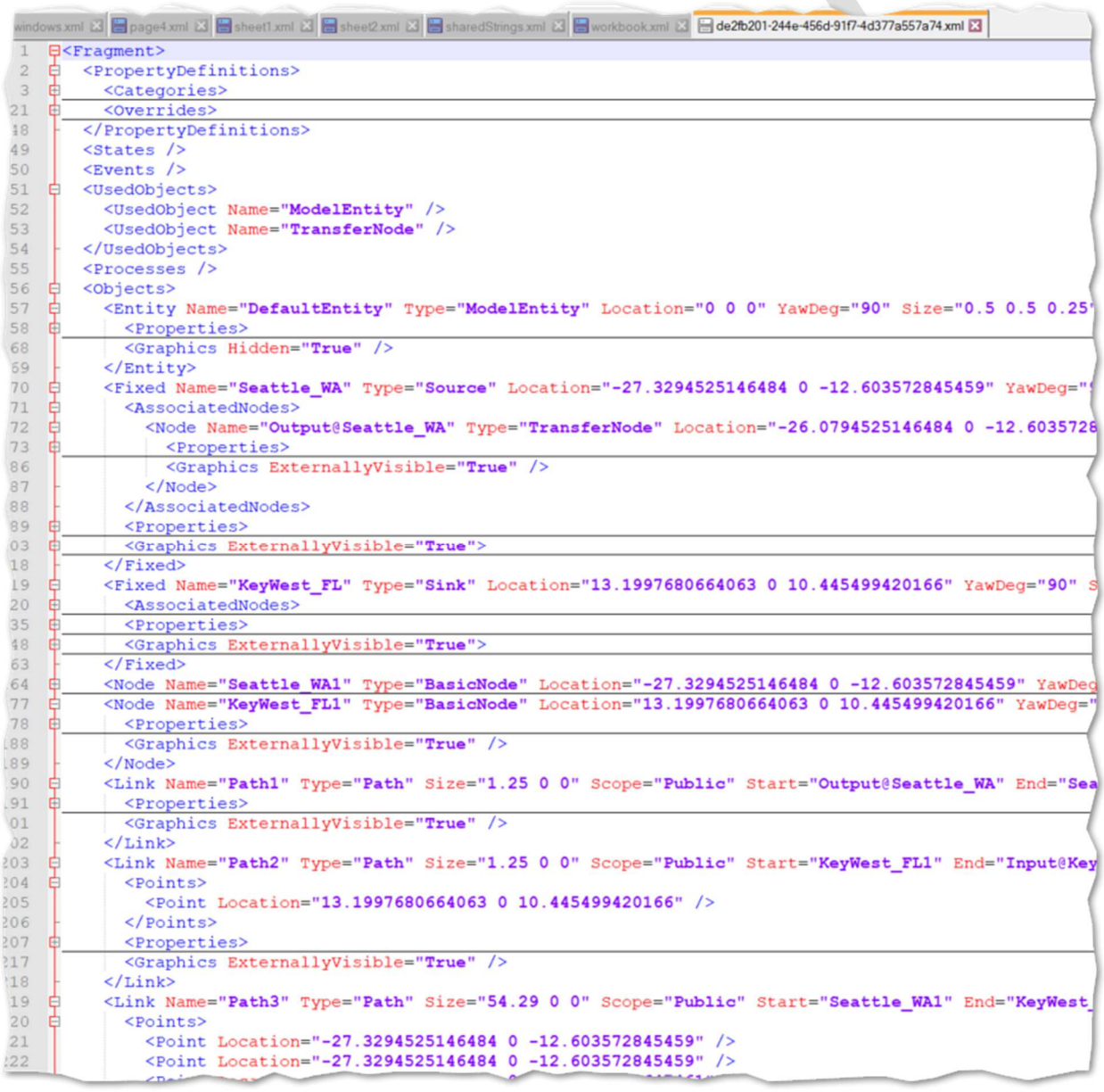


Simio
Forward Thinking

Appendix – Simio Project Container File (SPFX)

The Simio Project Container File (.SPFX) is stored in its own file, which is like an OPC structure (but as of this writing does *not* exactly conform to the OPC standard).

Objects contain the Simio objects. Within Objects are Simio Intelligent objects. In the example below, there is a Fixed object of Type “Source”, which contains a Node of type “TransferNode”.



Simio
Forward Thinking

Also seen are Link objects, which have ‘Start’ and ‘End’ attributes that reference Nodes. A Node can be an object, existing directly under Objects and can also be referenced by another object, as seen by the Output@Seattle_WA being nested under the ‘Source’ object Seattle_WA.

A large path is shown below, with its geometry being represented by Points within a Link (and the Link referencing the Nodes to which the Link is connecting).

```
    <Graphics ExternallyVisible="True" />
  </Link>
  <Link Name="Path3" Type="Path" Size="54.29 0 0" Scope="Public" Start="Seattle_WA1" End="KeyWest_FL1">
    <Points>
      <Point Location="-27.3294525146484 0 -12.603572845459" />
      <Point Location="-27.3294525146484 0 -12.603572845459" />
      <Point Location="-27.3280563354492 0 -12.6041488647461" />
      <Point Location="-27.3270721435547 0 -12.6030693054199" />
      <Point Location="-27.3180694580078 0 -12.5944519042969" />
      <Point Location="-17.4886169433594 0 -10.9811706542969" />
      <Point Location="-13.3983764648438 0 -10.824821472168" />
      <Point Location="-13.3903274536133 0 -10.8256416320801" />
      <Point Location="-12.4553070068359 0 -10.5785102844238" />
      <Point Location="-8.85404205322266 0 -9.68141174316406" />
      <Point Location="-8.85359954833984 0 -9.67145156860352" />
      <Point Location="-8.85971832275391 0 -9.64810943603516" />
      <Point Location="-8.63739776611328 0 -9.46989822387695" />
      <Point Location="-8.63349914550781 0 -9.46810150146484" />
      <Point Location="-1.77102661132813 0 -8.60879135131836" />
      <Point Location="0.430397033691406 0 -4.10707092285156" />
      <Point Location="0.429450988769531 0 -4.10641860961914" />
      <Point Location="0.427810668945313 0 -4.10412979125977" />
      <Point Location="0.427268981933594 0 -4.09970092773438" />
      <Point Location="4.85128021240234 0 -3.63618087768555" />
      <Point Location="4.85785675048828 0 -3.63059997558594" />
      <Point Location="6.06372833251953 0 -3.27571868896484" />
      <Point Location="6.00849914550781 0 -2.61581039428711" />
      <Point Location="6.00729370117188 0 -2.59963989257813" />
      <Point Location="9.78240203857422 0 -0.00403976440429688" />
      <Point Location="9.78826141357422 0 -0.00135040283203125" />
      <Point Location="11.1875305175781 0 2.04637145996094" />
      <Point Location="11.2874603271484 0 2.24657821655273" />
      <Point Location="12.9139022827148 0 6.14748954772949" />
      <Point Location="14.5263900756836 0 9.54521942138672" />
      <Point Location="14.5247802734375 0 9.55054092407227" />
      <Point Location="13.2020492553711 0 10.4480590820313" />
      <Point Location="13.2000274658203 0 10.4453392028809" />
      <Point Location="13.1997680664063 0 10.445499420166" />
      <Point Location="13.1997680664063 0 10.445499420166" />
    </Points>
    <Properties>
      <Graphics ExternallyVisible="True" />
    </Link>
  </Objects>
```