### **Introduction**

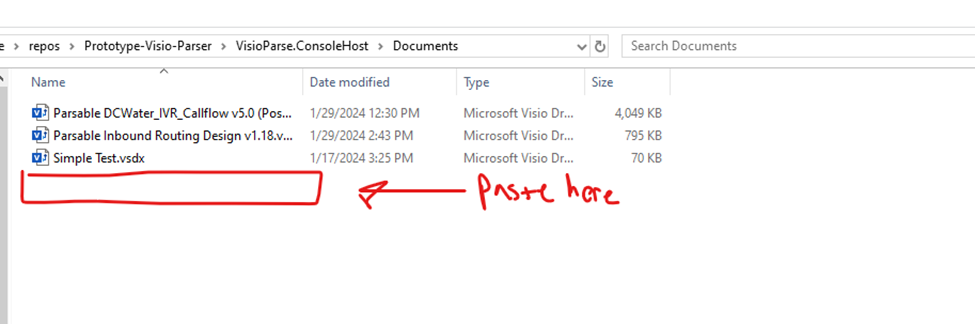
This report is intended to detail the usage of the QA Visio Parser program and enable it to best support a given Microsoft Visio file. It is intended to enhance the QA development experience by providing supporting information and assisting with test case creation.

### **Set-Up**

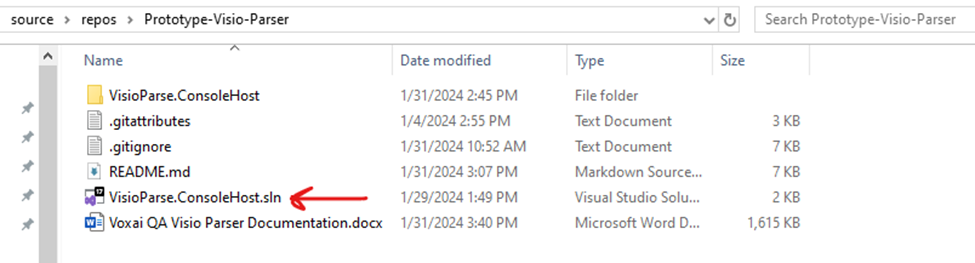
1. Ensure that Visual Studio with .NET support is already installed with a repository folder configured, then make sure that the folder for the project is contained inside of that repository folder.

Note: The default repository location for Visual Studio is C:\Users\{username}\source\repos, ensure that ZipTest.ConsoleHost is installed there.

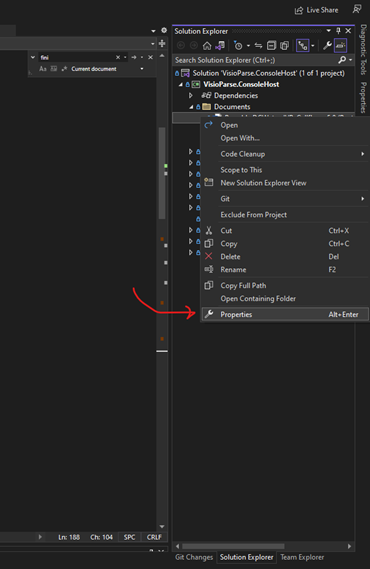
1. Open up the Prototype-Visio-Parser repository folder and then the VisioParse.ConsoleHost folder, the following Documents folder is where supporting documents should be placed, for example the Visio file desired to be parsed. Multiple Visio files can be placed in here, but the program will only run one at a time.



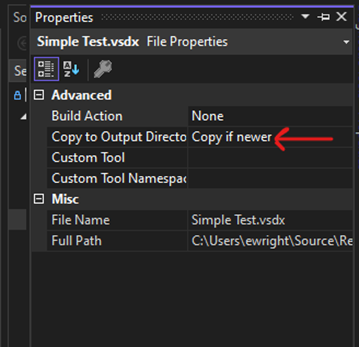
1. Return to the Prototype-Visio-Parser project folder and double click on the .sln file, it should open up the project in Visual Studio.



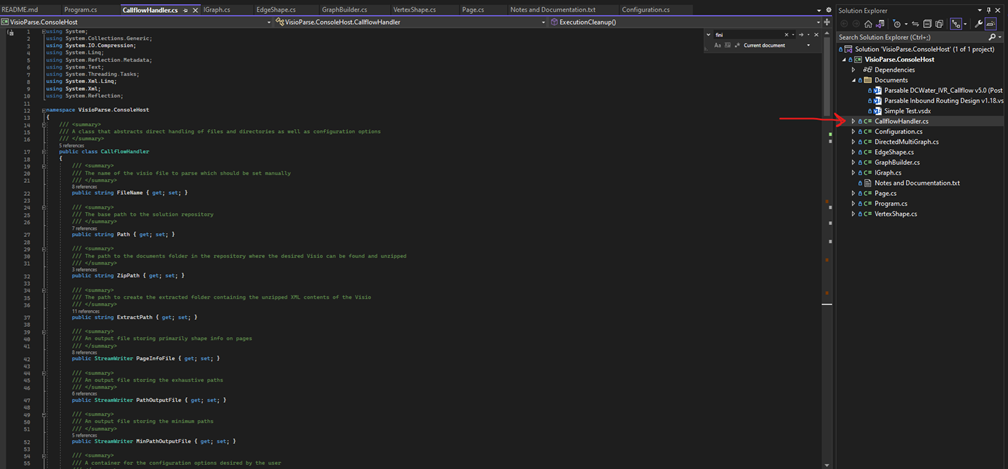
1. In the Solution Explorer tab on the right, open up the Documents folder, then right click on the added Visio and click “Properties”



1. Ensure that the file properties include “Copy if newer” for the “Copy to Output Directory” action

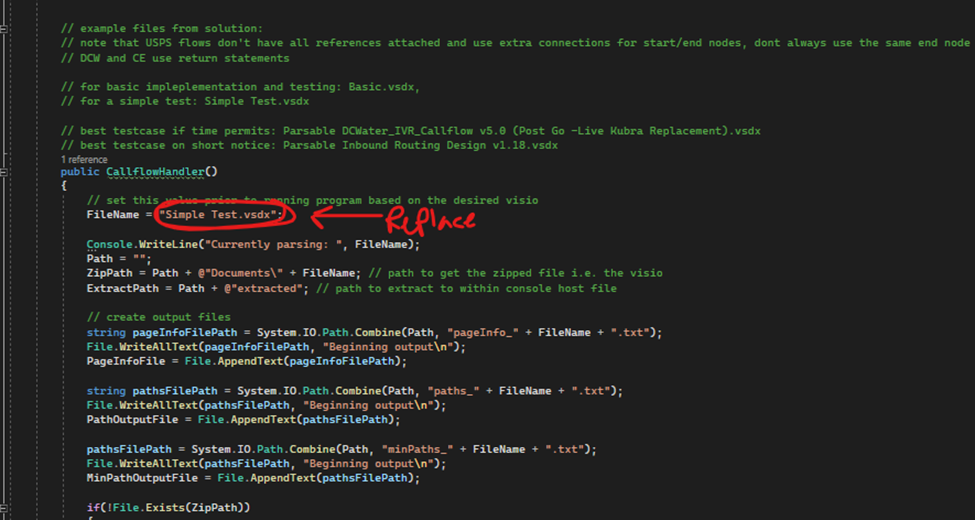


1. Return to the solution explorer and double click on the file named “CallflowHandler.cs” to open it up.



1. To parse the given Visio file, the CallflowHandler needs to know where to find the file. So scroll down to the CallflowHandler constructor and edit the following line defining the FileName as the name of the Visio file itself (including the .vsdx extension).

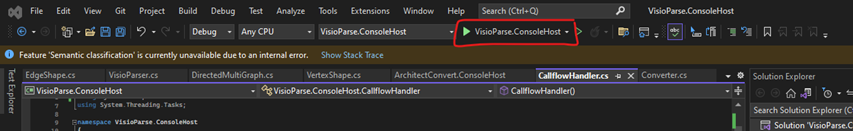
Note: These lines don’t need to be typed by hand, the file name can be copied (right click on the desired Visio file from the solution explorer, click rename, copy the text, then paste it into the FileName value)



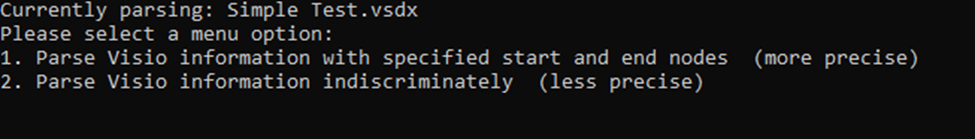
### **Parsing a Visio File**

The following instructions will guide a user through the program execution to select the right configuration settings and interpreting output. Note that the output will appear differently for each Visio due to the parsing of its information, so take caution when comparing the following steps as the output is produced from the provided example.

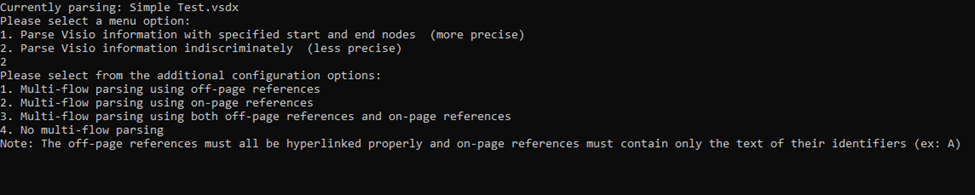
1. Click the execution button at the top to begin running the program



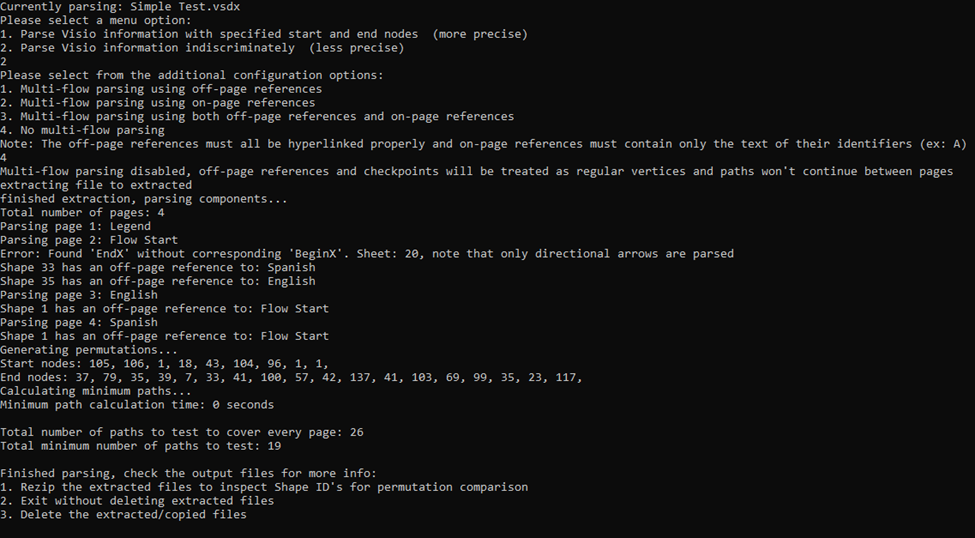
1. From the window popup, a list of menu choices will appear.. With no knowledge of the contained Visio data, it is safe to pick option 3. Type “3” and press enter.



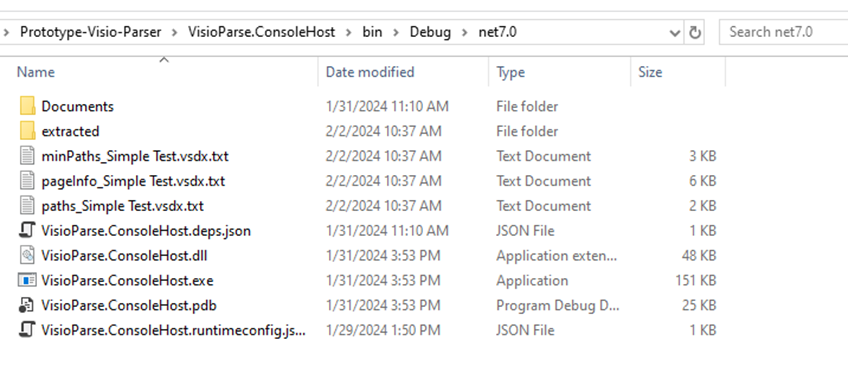
1. The second menu choice deals with reference parsing. For now, let the parser generate some simple internal data and we can consider this step later when we have more information. Select “4” to move on.



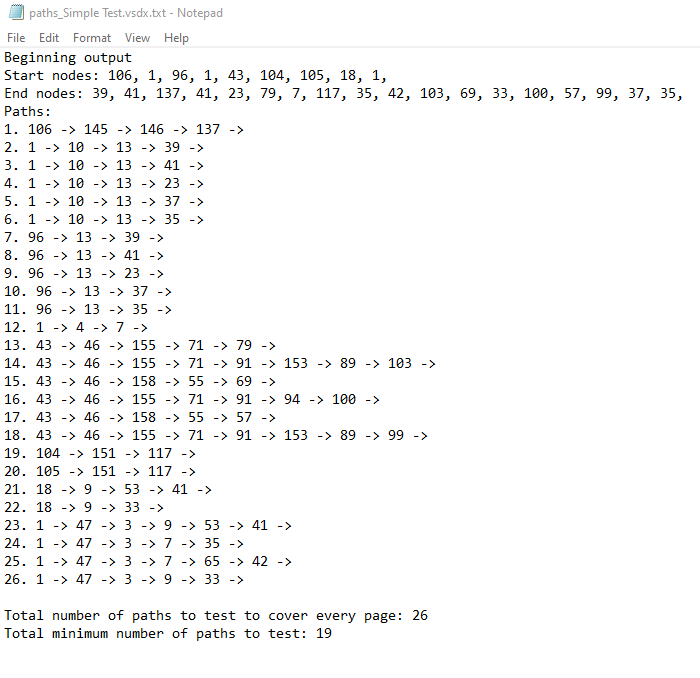
1. The program will parse the Visio and the console window will provide some benchmarks and logs. At the end of the parsing, we can see that the test for this example Visio produced 26 paths and a minimum number of paths to test as 19.



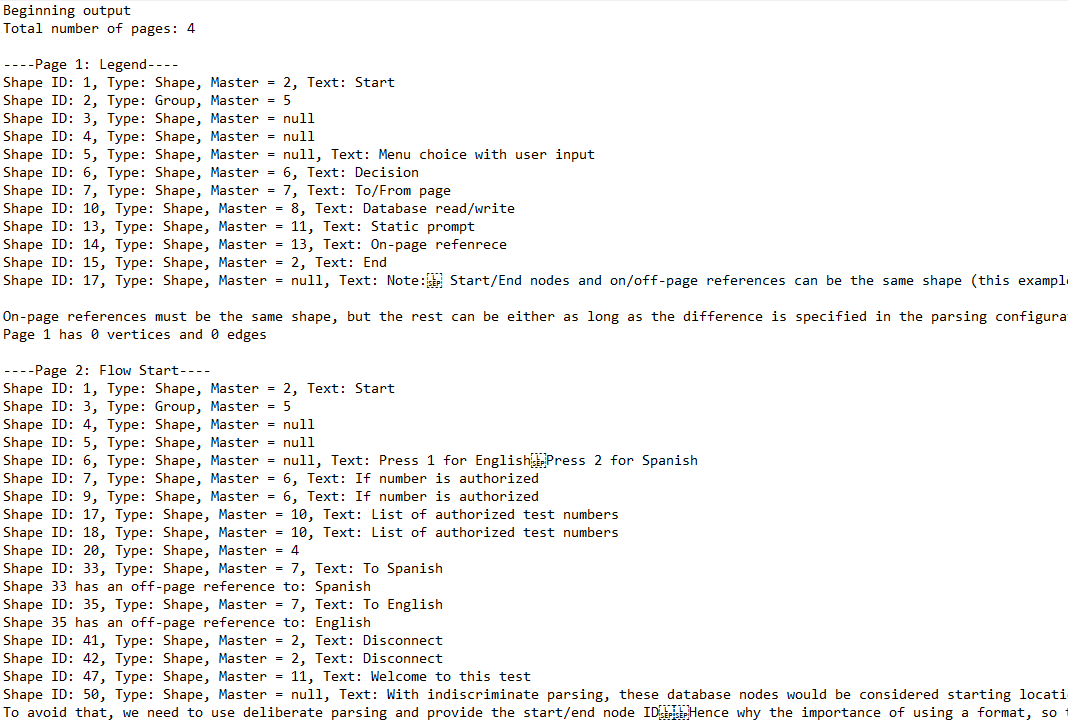
1. Next, reopen the project folder (\bin\net7.0\) and notice that there have been a few additions.
   1. In the paths file, we can see the details of each path generated by finding permutations between all possible start nodes to all possible end nodes. The paths are listed as the IDs for each shape, but are printed in text as well to be more readable and quickly interpretable.
   2. In the minPaths file, we can see the minimum paths that have been calculated from the total paths that cover every edge. For test case purposes, we can focus on these as they will cover every case to be tested, but our current version is an overestimate due to indiscriminate parsing, so we can leave it alone for now
   3. In the pageInfo file, we can see a breakdown of the internal information for each page, which can help interpret some of the paths as well as assist in extra configuration in later steps
   4. Furthermore, the folder named “extracted” contains the XML data that was extracted from the provided Visio and parsed, it can potentially be used in the subsequent step but can largely be ignored



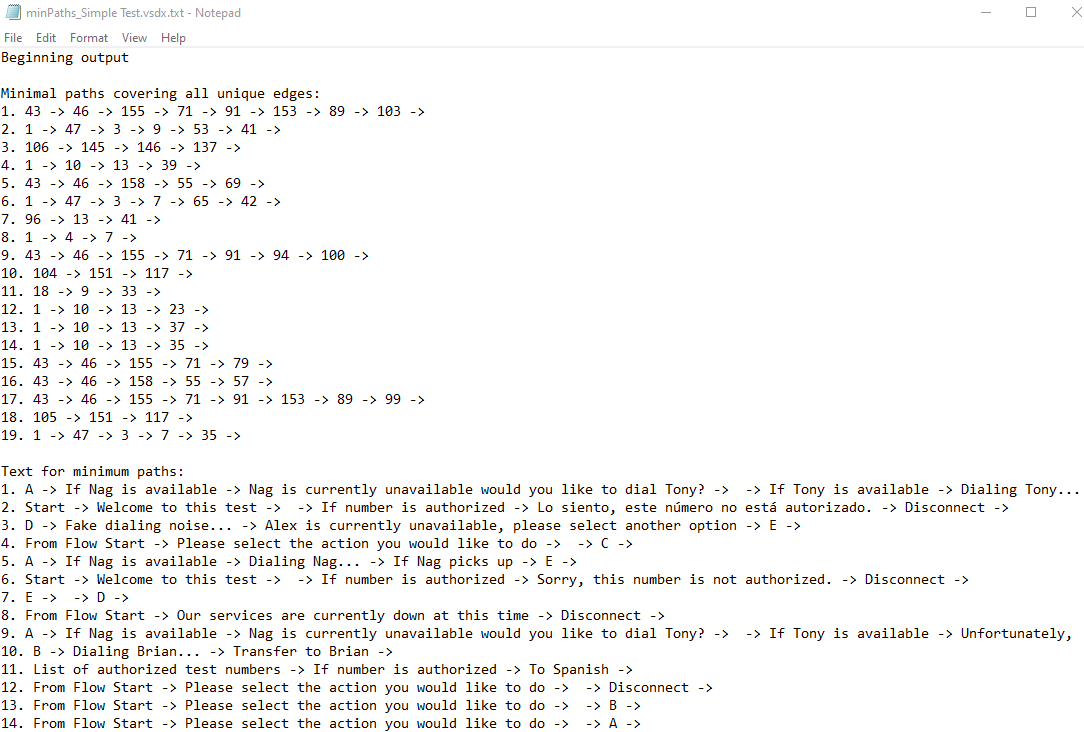
Example path file contents:



Example pageInfo file contents:

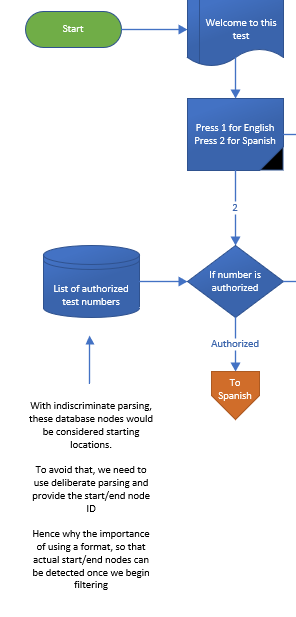


Example minPath file contents:



1. Back in the console window, more options are provided which deal with the extracted files. Note that these extracted files are copied from the Visio, not converted. This means that they can be safely deleted and the original Visio is still contained in the Documents folder with no modification
   1. The first option will allow for zipping the extracted files back up into a modified Visio, but with all text replaced as the Shape ID to help with additional visualization of the permutations. To do so, press “1” and then enter.
      1. Note that there is a subsequent option to delete this modified file as well, just press “d” and then enter. Anything else and program execution will stop and the file will be retained
   2. The second option leaves the system as-is and exits to allow the user to keep the XML content. Note that the XML text fields have been overwritten with the ID. Furthermore, the “extracted” file must either be removed from the directory or deleted manually to allow for the program to run again, it won’t overwrite previously extracted contents. To do so, press “2” and then enter.
   3. The third option handles the deletion for the user. It simply gets rid of the “extracted” file, the output files remain there for the user to view even after program execution is done. Furthermore, the output files are overwritten on each run so they don’t need to be cleared. To do so, press “3” and then enter.

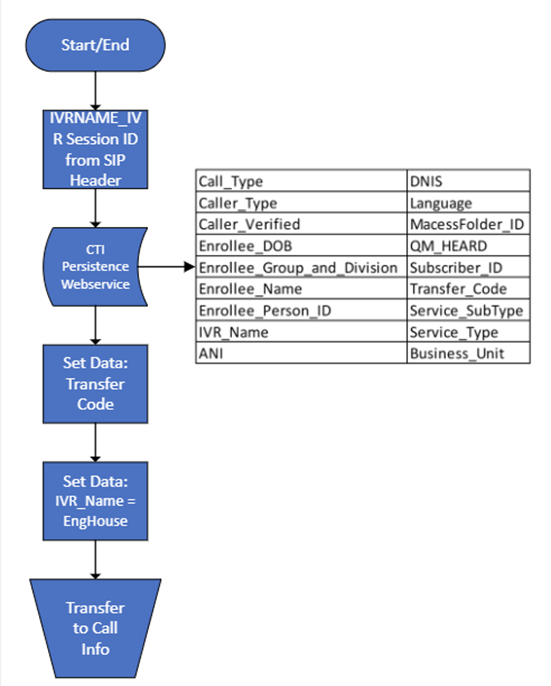
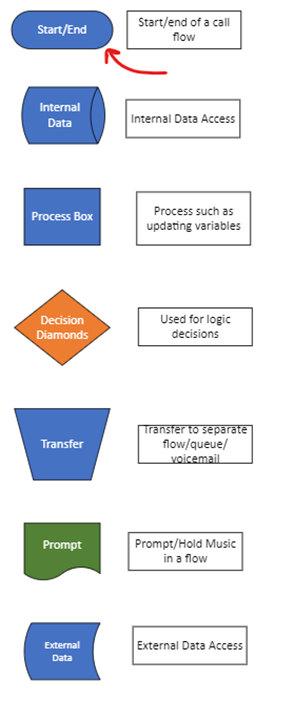
### **Modifying Parsing Settings**

For indiscriminate parsing, start nodes are chosen from nodes on the page which have outgoing edges but no incoming edges (with end nodes being the opposite). This works for most cases, but can cause some troubles with unintended nodes such as the following case in the used example showing how the table would be counted as a start node and leading to a large overestimation of paths in more complex files:  
   


#### **Note on Design Pattern Concerns**

As a result, we can be more specific in our parsing by permitting user-defined start and end nodes. Because of this, users are given some design freedom in their Visio choices, but try to keep starting and ending locations uniform across every page so that the configuration settings for the parsing can be as precise as possible. For example, the following Visio used in a different document uses a uniform shape for each start/end node. However, sometimes the author defines start nodes as “Start/End”, and there is no end node (flows should either end in a disconnect/end shape or an off-page reference to transfer to a flow that does end). In cases like this, there should be only one path, but instead we complicate things into one of two possibilities:

1. The user chooses to parse indiscriminately, whereupon the example page would return two paths (one from Start/End to the table, and one from Start/End to the bottom block labeled “Transfer to Call Info”).
2. The user chooses to parse precisely by providing the Start/End Master ID (which will be explained momentarily). In this case, no paths are found on this page because no end node can be found given the options.

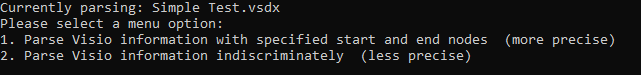


In either case, neither option will find the correct paths, because the Visio does not follow uniform constraints. Because of this, users must be careful in their design choices if they wish to use an automation tool. The choice to be more precise also requires an element of precision when designing to ensure the closest results, which means that users must clearly define where their flows begin and end, and stick to that denomination for the entirety of the file.

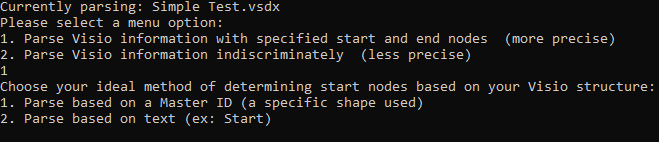
#### **How to Parse Precisely**

To enable the parser to find paths with constraints, we can use the following steps

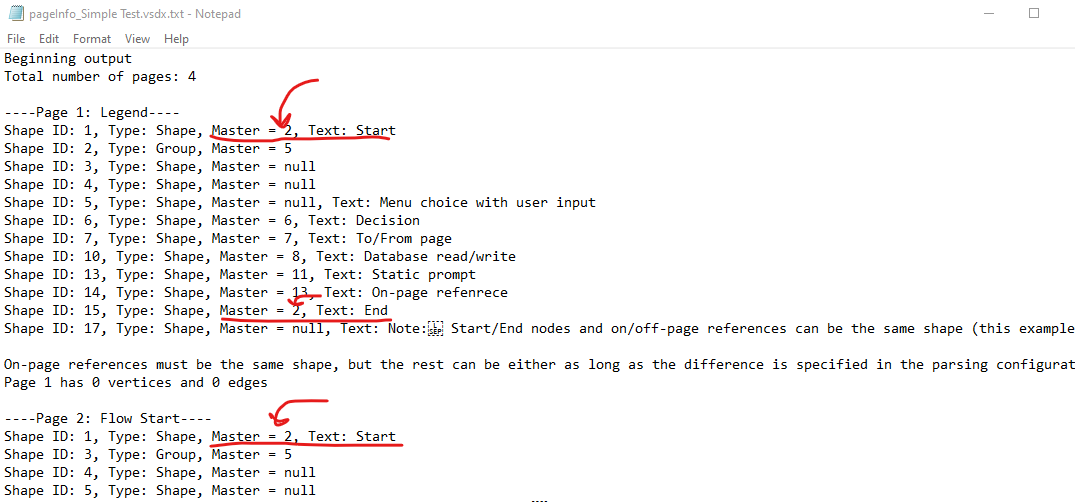
1. Click the execution button at the top to begin running the program
2. From the window popup, a list of menu choices will appear. To parse Visios with specified constraints, select option 2. Type “2” and press enter.



1. From the following menu, we can choose between defining nodes by Text or defining nodes by Master IDs. Enter “1” or “2” depending on which option is more consistent in the given Visio file.



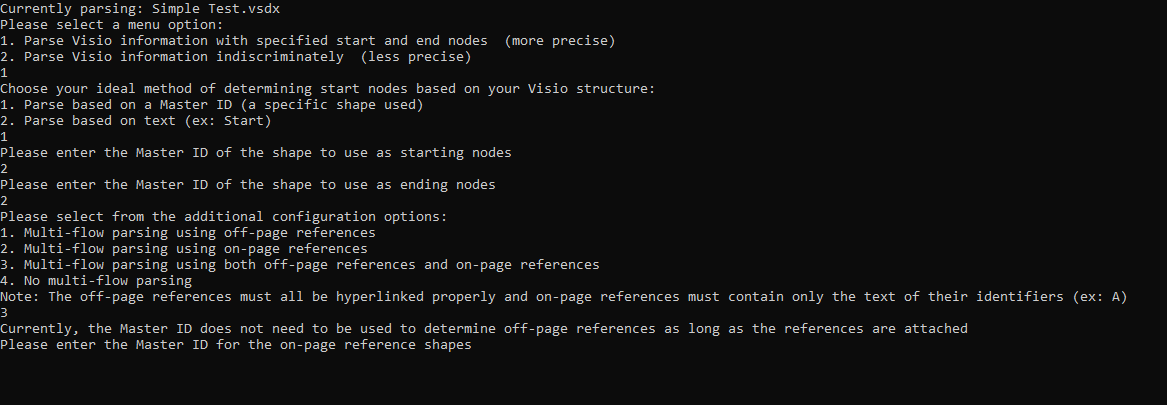
* 1. To parse by Master ID, reopen the pageInfo file from the indiscriminate parsing. If the given Visio includes a legend on the first page, then simply find the Master of the desired shapes to use. Then enter the start node Master ID followed by the end node Master ID (in this case, both the Start and End are “2”)



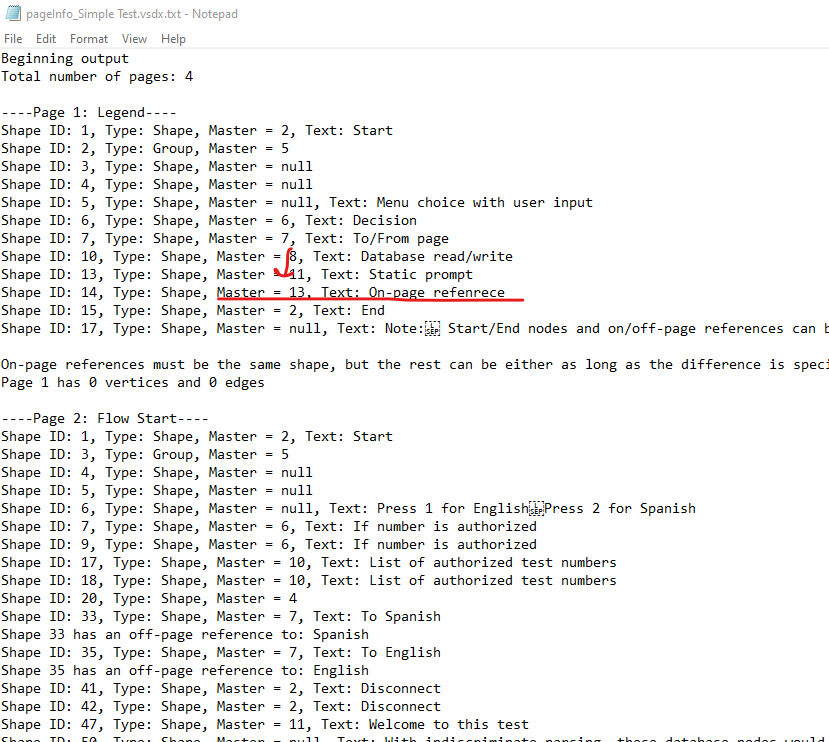
Note: if start and end nodes have the same Master ID, the program will still find the correct paths by interpreting their categories from the directed edges, so entering “2” and then “2” again is generally acceptable for flows which start and end using the same shape

* + 1. Alternatively, any instance of a start or end node that can be found in a page can be used, as long as the Master is consistent and all desired nodes adhere to the same ID

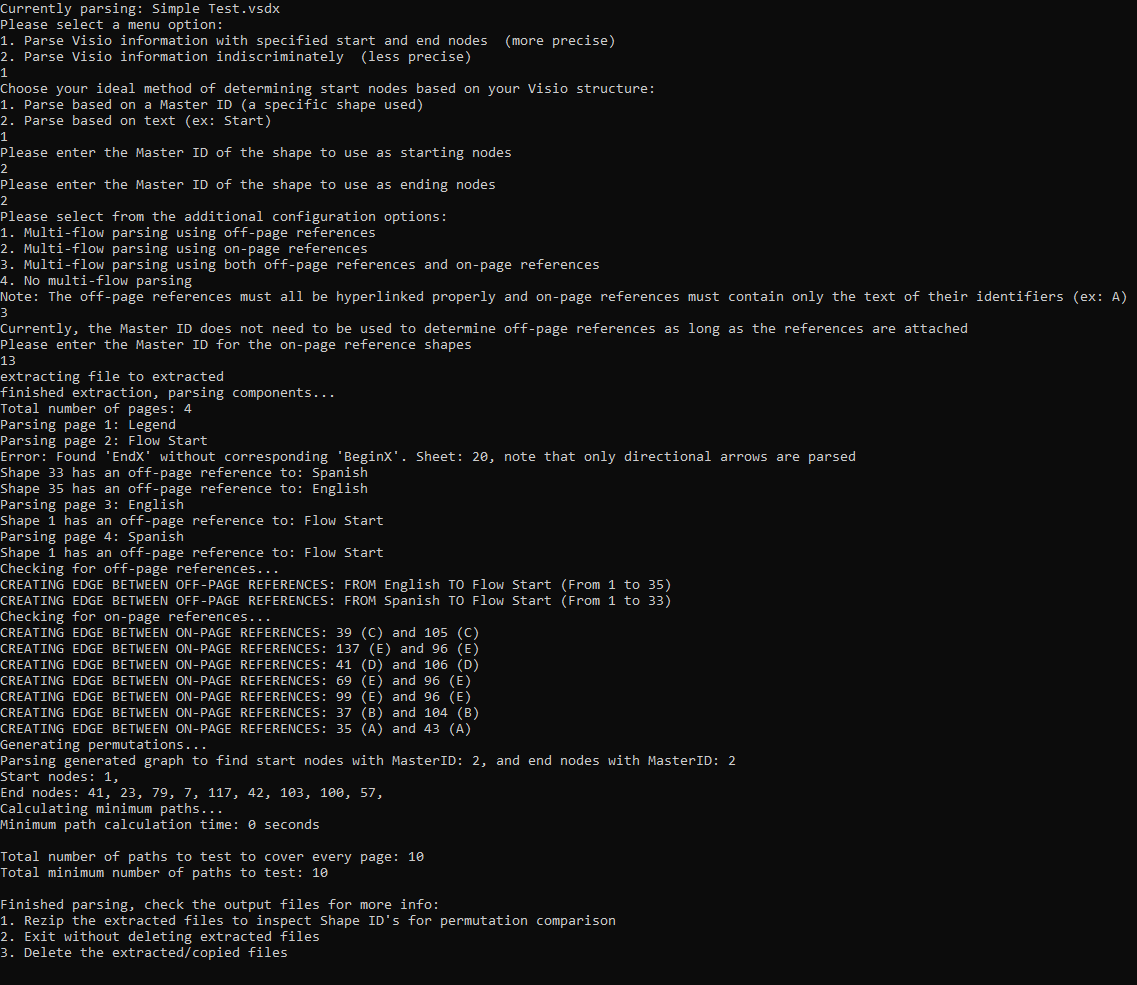
1. Afterwards, we have the following menu dealing with references to consider. This Visio uses both on and off page references, so we will select option 3:



* 1. Now, we need to enter the Master ID for the on-page references, which we can get the same as before by checking the previously generated pageInfo file (in this case, it is “13”):



1. If the parsing is successful, the output should look something similar to the following, with logs for connecting various on and off page references, as well as a likely reduction in start/end nodes:



Note: The paths and subsequently the minimum paths will likely change as a result of these changes unless the Visio is incredibly simple (This Visio went from 26 paths and 19 minimum paths to 10 paths and minimum paths). The reduction in start and end nodes will help reduce the amount of paths, but generally connecting references will increase the amount of paths. For Visios that have a large amount of pages all ending with multiple off-page references, this multiplicative difference can cause runtime to increase drastically, but take solace in the fact that the program results only need to be generated once to be recorded forever

1. Handle final execution logic by referring to Step 5 of “Parsing a Visio File”.

### **Interpreting Results and Identifying Issues**

As can be seen, the total path count is generally lower with added specification. However, in some cases, this can be primarily due to design styles being too inconsistent to accurately determine specified node locations. Some examples have been provided from a different Visio to help document the importance of sticking to a design:

1. On several pages, the specified design from the legend isn’t followed in some way. When parsing by text (using “Start” as the start node and “End” as the end node), we get the following output, which shows the program suddenly is unable to find the paths for these pages, each due to reasons regarding design issues

A screenshot of a computer

Description automatically generated  
  
 Let’s take a look at some of these reasons specifically:

* 1. On Pages 2 and 3, the end node shape and “End” text is not used, which means that neither specified parsing method will be able to detect any paths in the entirety of both flows. While these are indeed intended to be transfers to other pages, they do not include the link that is necessary to traverse to that page, and as such the parser will not continue the flow:

A diagram of a computer code

Description automatically generated

A diagram of a data flow

Description automatically generated

* 1. On Pages 4 and 11, the end node is called “Disconnect” or something else instead of “End”. Fortunately, it is still using the same Master shape as the defined end node, but it means that parsing by text would not work for this flow because the end node would not be located:

A diagram of a flowchart

Description automatically generated

1. So at a first glance, when this Visio generates 5184 test cases, it seem accurate. However, the reality is that every page (save for page 9, which happens to be the largest) has a design formatting issue, causing the true number of paths to be incalculable when parsing by text because the legend is not followed - which can be seen by the lack of test cases for every other page:

#### **Design Debugging Techniques**

Another useful technique to use while debugging the design of a flowchart is the ID mapping functionality. For example, after we extract the files and remap it back into a Visio file, we can observe the paths of page 10 which should have only one start node (Shape 1) and one end node (Shape 171):  
 A number on a white background

Description automatically generated  
After rezipping up our modified Visio for inspection, we can see that this page has some faulty connections which cause unintended start nodes when using indiscriminate parsing and also prevents the path from the true start node from ever reaching the true end node. In the following image, it’s clear to see that there is an issue with the connection prior to Shape 140, which is causing the connection to fail, subsequently promoting Shape 140 to an inpromptu starting node:

A diagram of a work flow

Description automatically generated

Taking a look in the original Visio, we can see that the connection was never attached:

A diagram of a flowchart

Description automatically generated

Furthermore, because Shape 17 is labeled as a start node in the indiscriminate parsing, we know that this arrow design isn’t actually connecting to the node. If the edge is not anchored to the node, then it will not be considered connected, which causes the parsing tool to discard the corresponding edge entirely.

A diagram of a diagram

Description automatically generated

This technique can assist with formatting the Visio in such a way that transfers it from being just human-readable into machine-readable. Be sure to analyze the results of the parsing to ensure that the Visio is designed in an optimal way using best practices.

After editing the Visio to patch up design inconsistencies and adding explicit start and end nodes, we reach this output:

A screenshot of a computer

Description automatically generated

23816 total paths using the specified Master ID parsing, far from the original number of 5375 (using Master ID vs 5184 using Text), proving the importance of ensuring that the Visio designs are completed before they are used for test generation.

However, this number is an incredibly large overestimation representing every single possible path which could be taken. Rather, the minimum number of paths may be sufficient in most cases, which simply calculates the minimum number of paths needed to traverse all edges and therefore test every case. The minimum set of paths generated by the program are therefore not perfect, but ideal considering the massive resource increase needed for exhaustive testing of every possible scenario. The minimum paths themselves are located within the paths file along with the exhaustive paths in the minPaths file.

### **Ideal Design Practices**

A large part of ensuring that the tool works as expected and has accurate results is ensuring that design practices are kept consistent and preferably up to universal standards. Even when parsing indiscriminately, there are various features in the application which make assumptions about the overall flow and design of the Visio file, which only become extrapolated when attempting to parse precisely. For that reason, it is important that the Visio files provided to the application are constructed such that these assumptions are met in order to maximize the usefulness of the results. Here are some notes to take into consideration when designing a Visio intended to be parsed:

General Tips:

* Don't make dense flows, make long ones (i.e. shapes shouldn't have paragraphs of text, the more concise the file is the better it will be for readability later, split into multiple shapes if necessary)
* Make sure that all connections are actually connected
* Non-directed connections won't be used for test case calculation, they can be used to label things for human-readable portions but not for parsing
* Page names must be unique in order to differentiate them for off-page references

Start/End formatting:

* stick to one format for start and end nodes, as well as on-page and off-page references
* To use specified start and end nodes, ensure that start nodes have no edges pointing in to them and end nodes have no edges pointing out of them
* For start and end nodes, try to use a basic shape to ensure that it will have a master ID
* For start nodes, don't attach extra connections for extra visualization (i.e. a picture of a person to represent an agent pointing into a start node)
* start nodes should not have any incoming connections, end nodes should not have any outgoing connections, else they will not be recognized properly

Reference formatting:

* Avoid using connections in a legend page or other non-flow page to prevent unnecessary paths from being generated when parsing indiscriminately
* Multiple on-page references must be used appropriately, avoid using more than one pair of start/end on-page references with the same identifer on the same page
* However, multiple on-page end nodes can refer to a single on-page start node and vice-versa, simply don't use the same identifier for more than one pair
* Don't use the on-page reference shape as off-page references, the paths will not traverse through that shape.
* Off-page references must have a matching off-page reference that points back to it
* If there is not a matching off-page reference to receive the flow, the program will attempt to create an edge to the start node on that page instead, but if there is more than one then the edge placement is not possible to determine
* Therefore, this step can be ignored as long as the page only has one start node, else the reference start will need to be specified by a receiving off-page reference
* Make sure that all off-page references have their links attached properly or else the parser can't build a multi-flow path

### **Conclusion**

In conclusion, the Visio parser can help determine test cases for QA purposes and provide various useful information, but it requires careful consideration of design to ensure that the information is accurate. Please use caution both when building the Visio and when interpreting the results.