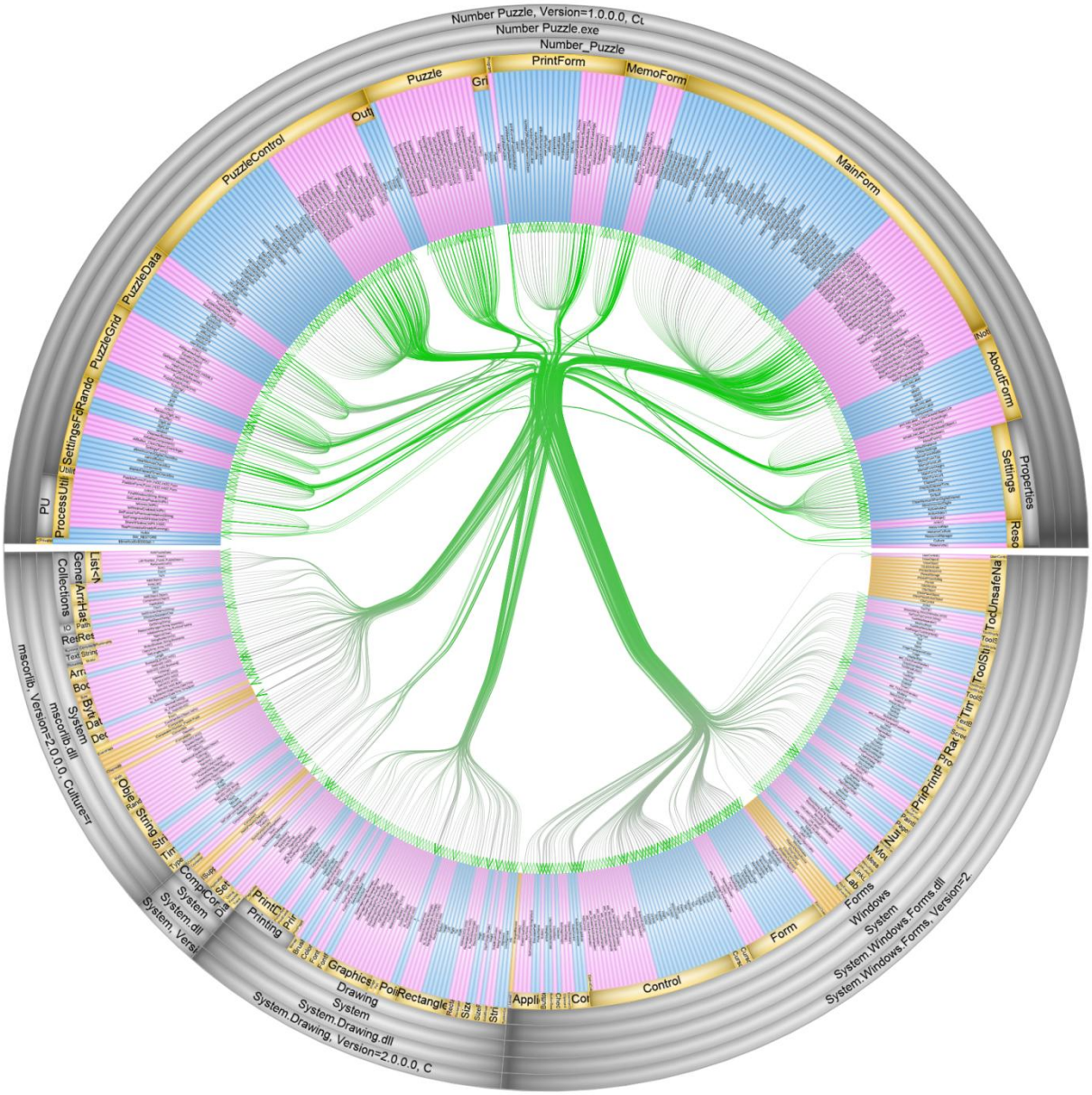
SolidSource

Solid Software Xplorer

User Manual

for version 1.4, September 2009



www.solidsourceit.com

Contents

1 Introduction

Solid Software Xplorer (SolidSX) is an application that helps you explore, analyze, and understand the source code of (large) software systems. A good understanding of a software system supports decisions for code refactoring, removing code clones, identification of design patterns, and debugging, to name a few examples. As such, SolidSX is a useful tool for software developers, designers, and architects.

Instead of displaying the code textually, SolidSX represents the software system graphically. The human visual system is known to have a high data bandwidth and throughput, and SolidSX uses this property to convey large amounts of information to the user by creating novel, high quality visualizations of the software. In that sense, SolidSX is better suited to gain code insight than classical text-based approaches.

SolidSX visualizes the following aspects of the software system:

- Source code **elements**, such as namespaces, classes, and methods.
- The **element hierarchy**, or in other words the nesting structure of elements. Typically, namespaces contain classes, classes contain methods, etc., but other hierarchies can be thought of.
- **Relations**. Elements have various types of relations between them. For example, a method that calls another method has a calling relation. Another example is an inheritance relation from one class to another.
- SolidSX visualizes **metrics** on both elements and relations. Example metrics are the number of local variables, lines of code, and code complexity.

SolidSX makes (large) datasets insightful by combining these aspects into a single visualization and by facilitating easy inspection and navigation through a user-friendly interface. Hardware-accelerated OpenGL graphics are used to ensure high visual quality, fast performance, and smooth navigation.

Some of the features of SolidSX are:

- Visualizes large software systems using state-of-the-art visualization techniques
- Uses hardware-accelerated OpenGL graphics for high quality visualizations and smooth navigation.
- Provides a clear, minimal interface for high ease of use.
- Extracts information from Microsoft .NET projects, written in any of the .NET languages, such as C#, Visual Basic, Visual C++ (managed), and Silverlight. The source code is not needed: just import your executable or DLL.

- Can be used stand-alone to analyze any compiled .NET assembly, but can also be fully integrated with Microsoft Visual Studio as a plug-in. When used as a plug-in you can navigate between source code and visualization while you are coding, making SolidSX a convenient developer tool.
- SolidSX is not limited to exploring software systems, but can be used to visualize other dependency data you might have as well. If you want to know more about importing your own data, see Section 3.6.

2 Installation

SolidSX will run on most recent computer systems. SolidSX requires:

- Operating system: Microsoft Windows 2000, NT, XP or Vista (32 bit).
- Graphics card: OpenGL 1.1 compliant card, resolution of 1024 x 768 minimum, 1280 x 1024 or higher advised.

To use the .NET import functionality, you need to have the Microsoft .NET framework installed. However, SolidSX itself does not need the .NET framework.

The installation is straightforward. Download and execute the setup executable from the SolidSource website (http://www.solidsourceit.com).

After installation, you need to select a license file. We provide free evaluation and educational licenses on our website that will be sent to you by e-mail. If the license is expired, you can continue working with SolidSX, but a limit is put on the number of elements that a dataset may contain.

There are currently two editions of SolidSX:

- SolidSX **Lite** edition contains all the basic functionality needed for importing and visualizing .NET assemblies.
- SolidSX **Standard** edition has all the functionality of the Lite edition, but adds the following functionality:
  - The .NET importer generates several extra attributes, such as complexity and cast count.
  - Making high-resolution screenshots (up to 3000x3000 pixels) of your system
  - Loading and saving visualization states, for quickly accessing often-used views on the software system
  - Importing XML datasets that you generated yourself

Visit http://www.solidsourceit.com for more information.

3 Working with SolidSX

Working with SolidSX involves the following aspects:

- SolidSX datasets
- The user interface
- Navigation

We discuss these aspects in the sections below. If you want to get started immediately, you can follow the tutorial from Section 4.

3.1 SolidSX datasets

SolidSX visualizes Solid Software Xplorer **datasets**. You can obtain a dataset in three ways:

- Use one of the **importers** that come with SolidSX. There is currently one importer available for .NET assemblies. More converters will be available in upcoming versions.
- If no converter exists (yet) for your programming language or development environment, you can create a dataset yourself. The dataset has an XML format that is explained in Section 3.5.
- You can also open one of the example datasets that comes with SolidSX. This is useful to get acquainted with SolidSX. Example datasets can be found in the folder “examples” in the folder where you installed SolidSX.

A SolidSX dataset contains information about elements, hierarchies, and relations. A relation is defined between two elements and has a direction.

Each code element in a hierarchy has a parent element and has zero or more child elements. For example, the children of a class are its methods and fields, and the parent of a class is the file it is defined in. A dataset might define multiple hierarchies. SolidSX visualizes one hierarchy at a time, but allows you to quickly switch between hierarchies.

A dataset also defines directed relations between source code elements. A “call” relation, for instance, typically connects two methods and indicates that the first method calls the second method. Other types of relations are inheritance and field-access relations.

A dataset can further define a variable number of attributes on the elements and relations. Each element and relation has a set of key-value pairs, where the key is a string denoting the attribute and the value denoting its value. The value of an attribute can be a string, number or Boolean. Datasets typically define at least the “name” and “type” attributes for code elements, where the values of type will be “class”, “method”, etc. The creator of the dataset can define additional attributes, such as “parameter count” for methods. Relations typically have a “type” attribute, where the value of type is “call”, “field access”, or “inheritance” for example. Again, additional attributes can be defined. Section 3.5 lists the attributes that the .NET importer generates.

3.2 User interface

The user interface of SolidSX consists of one main visualization window and several inspection windows (see Figure 1). These inspection windows can be 'docked' into the main window or made invisible by closing them. At any time they can be opened again from the View menu. We now discuss the user interface in more detail.

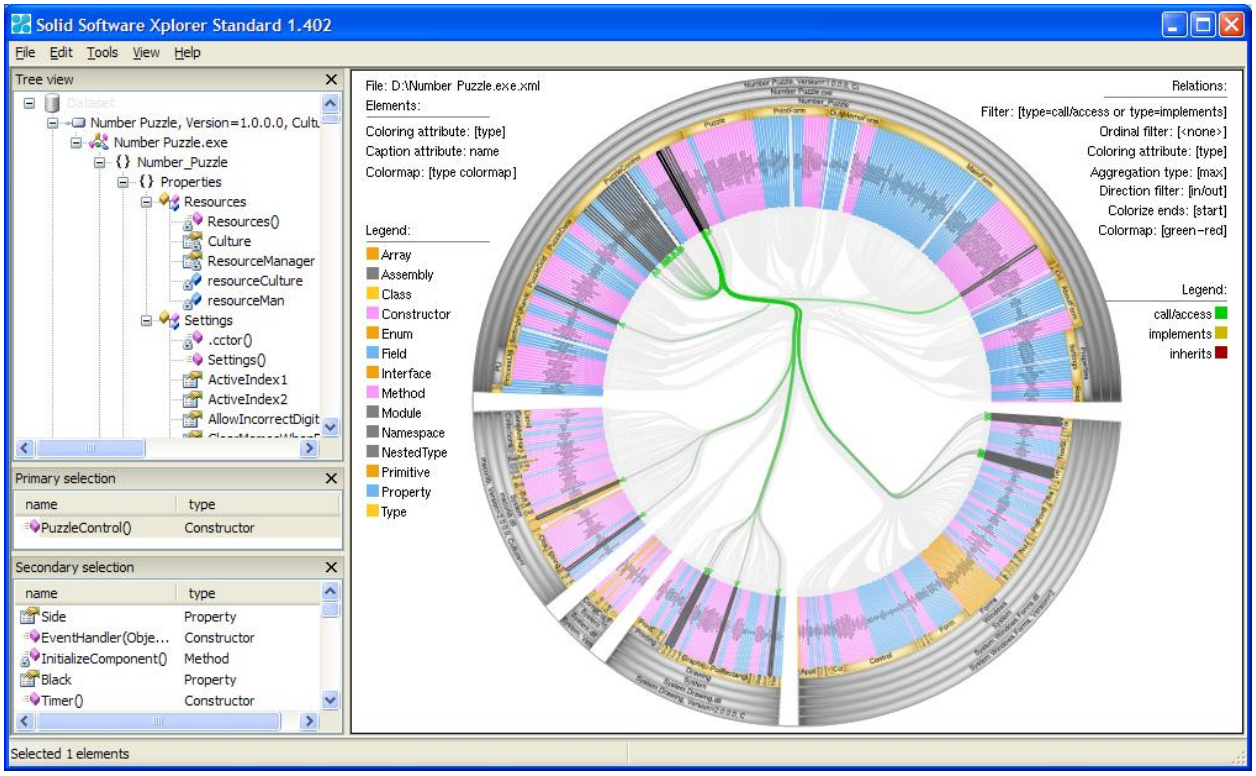


Figure 1: The user interface of SolidSX. The radial view is on the right, dockable inspector windows on the left. The legend is shown on the left and right side of the radial view.

The **radial view** is the main view of SolidSX. The radial view visualizes elements as (curved) boxes that are placed in concentric rings and visualizes relations as curved arrows in the center. The nesting of the boxes reflects the hierarchical structure of the elements: the outer ring contains top-level elements of the hierarchy, the lower level elements are found when going toward the center of the circle. The circle is divided in two halves: the top half and the bottom half. The top half contains elements from the user code, whereas the bottom half contains the external libraries on which the user code depends.

The **legend** is located at left and right side of the radial view and indicates how the current visualization should be interpreted, such as which attributes are used for coloring elements and relations, which relations are shown, and what color maps are used. The visualization parameters can be changed by interacting with the legend.

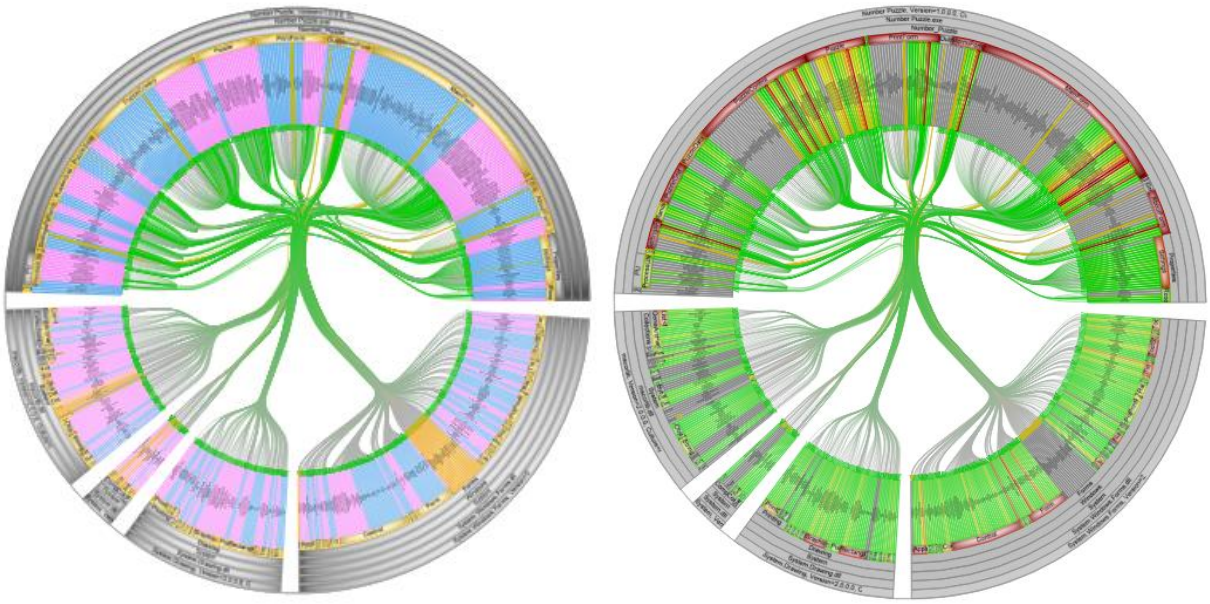


Figure 2: The radial view showing elements in the concentric rings and relations in the center. On the left: element color encodes element type, on the right: color encodes code size.

Each element has a **color** that encodes the value of one of its attributes. The attribute that is used for coloring is called the "element coloring attribute". The coloring attribute can be changed by the user from the legend. Each element has cushion-like shading, but lacks shading when the element does not have the coloring attribute (for example, a box representing a class element will not have shading when "parameter count" is the coloring attribute, as only methods have such an attribute). Figure 2 shows an example. On the left-hand side, the color encodes the element type: namespaces are gray, classes are yellow, methods are magenta and property/fields are blue. On the right-hand side, the color encodes code size, where red indicates a large, and green a small size. In this image, fields and namespaces have no shading, as they do not have the code size attribute.

Each element has a text label applied to it that shows the attribute "name". Naturally, the more elements are displayed, the less screen space is available for each element, which might make labels too small to read when many elements are displayed. You can inspect the label of an element by brushing it with the mouse cursor. A **tooltip** is displayed that clearly shows the values of the labeling and coloring attribute of the element. Alternatively, the **attribute inspector** can be used (F1) to inspect the element's name and other attributes.

The **tree view** provides another view on the element hierarchy, using the familiar tree control. The tree view only shows elements, no relations. Nodes can be collapsed and expanded from the tree view by clicking the plus and minus signs to the left of them.

Collapsing and expanding elements are important operations in SolidSX. A collapsed element does not show its children in the radial and tree view, so that the element takes up less screen space. When viewing a large dataset, it is not feasible to visualize all elements at once, so some elements need to be collapsed, for example of a sub-system which you are not interested in. Furthermore, by collapsing and expanding you can control the level-of-detail at which you view the system, for example by collapsing all class elements. Figure 3 shows two other collapsed states of the same hierarchy as shown in Figure 2.

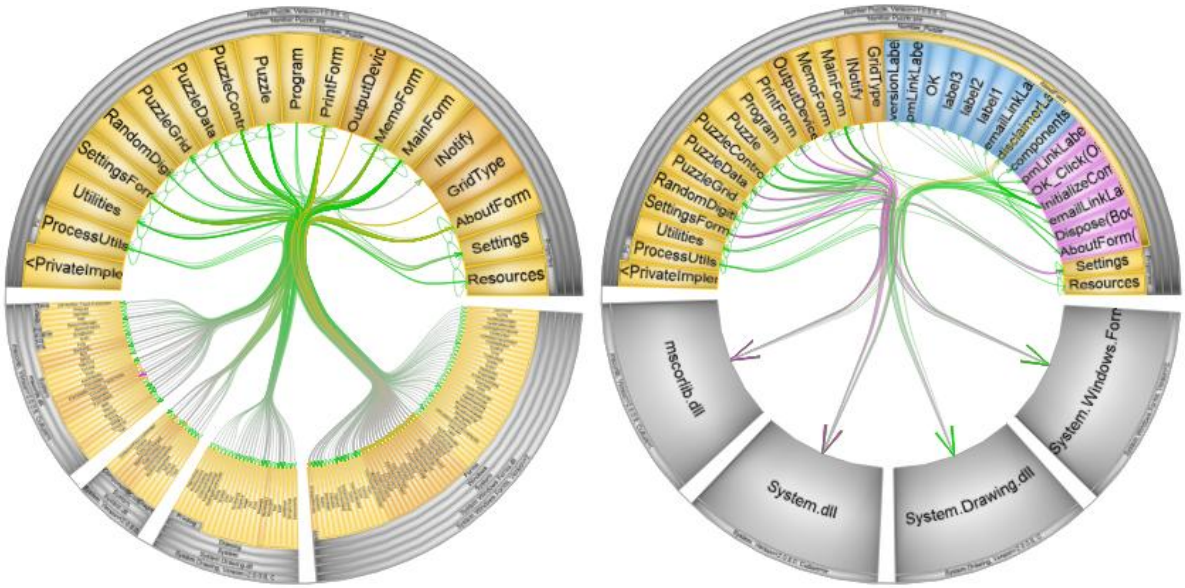


Figure 3: Two different collapsed states of the same dataset.

3.2.1 Relations

Relations form a major part of the visualization and are drawn in the center of the radial view. A relation between two elements is drawn as a curved arrow. For example, one method calling another is shown as an arrow pointing from the caller to the callee.

It is important to note that a single arrow can represent more than one relation. This is the case when the parent or another ancestor element of an arrow's end element is collapsed. Because the descendant elements of a collapsed element are not shown, the relations to these descendants need to be **aggregated**. This means that all relations from or to any of the descendants of the collapsed element, go to the collapsed element instead. For example, when collapsing a class element, all call relations from or to any of its methods will be aggregated and will end at the class element instead. Due to this collapsing operation, a single arrow in the radial view might represent many relations at once. To name an extreme case, by collapsing the top-level element of a system, only one element remains visible and only one arrow is drawn, namely a self-loop on the top-level element.

Arrow heads are drawn to indicate the direction of the relation(s) it contains. If the arrow represents only one relation, a single arrow head is drawn at the ending element. If the arrow represents multiple relations, two arrow heads are drawn if the arrow contains at least one relation in each direction.

As with elements, the **color of an arrow** encodes an attribute of the relation. The "relation coloring attribute" determines what attribute is used for coloring the arrow, and can be changed from the legend. Because an arrow can represent more than one relation at once, each relation within the arrow contributes its own color. If there are at least two different colors, the arrow gets a magenta color.

In addition to the arrow heads, the direction of the arrow is also encoded by making the arrow gray near the ending element. Whether the start or the end of an arrow is gray depends on a setting in the legend. If the arrow is bi-directional or if the setting is set to "start/end", none of the ends is gray. The encoding of direction in the color has some benefits over using arrows, especially when many arrows are drawn.

You might wonder why the **arrows are curved.** This is done to avoid the visual clutter that you would get when using straight arrows. SolidSX uses the hierarchy of the elements to curve the arrows, effectively "bundling" the relations that have hierarchically related targets[1]. Simply put, the relations of an element A are bundled with the relations of an element B by the common ancestor of A and B. Note that when there is no strong hierarchy, there will be limited bundling as a result. This indicates a weak modularity of the software system, and might be an indication of a poor design.

The bundling strength can be controlled from the settings window (F4), although it is not necessary to change the default setting. Figure 4 shows the advantage of bundling relations (left image) when compared to using straight lines between elements (right image). Bundling provides a more global view on the relations: the modularity of the system is reflected by the arrows. Furthermore, bundling prevents visual clutter.

[1] This technique of bundling relations was first described in the academic paper: "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data". Danny Holten. IEEE Transactions on Visualization and Computer Graphics (TVCG; Proceedings of Vis/InfoVis 2006), Vol. 12, No. 5, Pages 741 - 748, 2006.

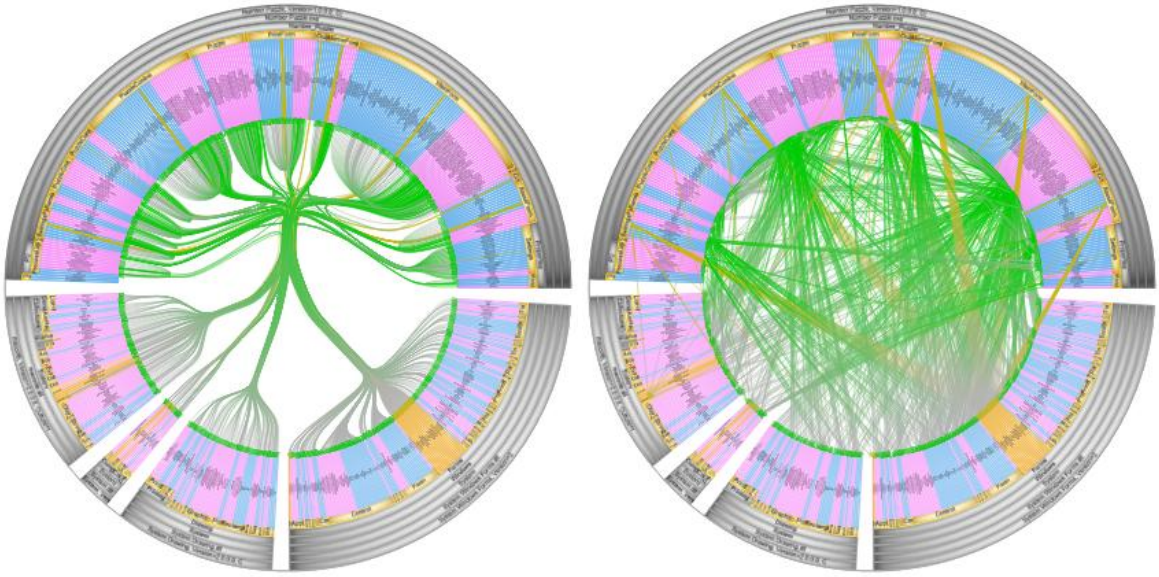


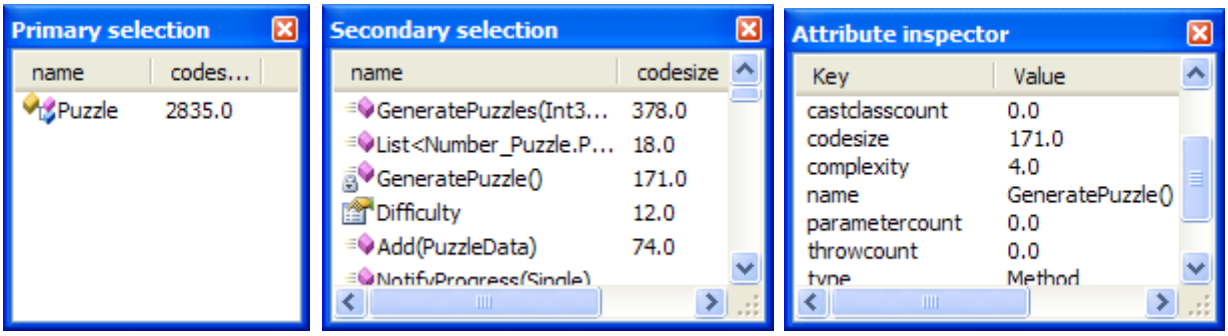
Figure 4: Bundled (left) versus non-bundled relations (right)

3.2.2 Inspector windows

So far, we have discussed the radial view and tree view. However, SolidSX provides some additional windows that enable fast and easy inspection of data.

The **attribute inspector** (F1) allows you to inspect all the attributes of the element or relation that you are currently brushing with the mouse.

The primary and secondary **selection windows** (F2 and F3) allow you to inspect the current primary and secondary selection. The primary selection consists of elements that are selected by the user (typically done by left-clicking them). The secondary selection is computed automatically by SolidSX and consists of the elements that are connected to (that is, have relations with) the primary selection. These selection windows also show attributes of the selected elements and can be used to sort them by clicking on the respective column.



The **settings** window allows you to set parameters of the visualization that are less often used than those in the legend. The settings window can be shown by pressing F4 or from the menu bar.

The log view stores informational messages, warnings, and errors that might have occurred. The log is also stored as “log.txt” in the application folder.

3.3 Navigation

Fast and easy inspection and navigation is an important feature of SolidSX. SolidSX offers a large number of navigation actions: brushing, selection, collapsing and expanding, and filtering, to name a few. We discuss these operations below.

3.3.1 Brushing

When **brushing an element** in the radial view (placing the mouse cursor on it) the element gets a thick outline and a tooltip is shown at the mouse position. The first line of the tooltip shows the name of the element. The second line shows the value of the coloring attribute of the element. If you want to see the other attributes, you have to open the attribute inspector (F1). Note that you can control the font size of the tooltip from the settings.

Furthermore, when brushing an element, only the relations that are incident to the brushed element are shown; the others are made gray. Whether elements that are connected to the brushed element by incoming or outgoing relations are shown, is determined by the “direction filter” found in the legend.

Most keyboard shortcuts (see Section 3.4) also work when brushing an element. You do not need to select the element first, which saves you a mouse click but more importantly: it leaves the current selection intact. Brushing elements also works in the tree view and primary and secondary selection windows.

When **brushing an arrow** you have to hold ALT to show the tooltip. The tooltip shows the number of relations that the arrow represents and the direction of the relations. Because arrows can overlap, it is possible to brush multiple arrows at once. When brushing multiple arrows, this is indicated by the text in the tooltip. When brushing **an aggregated arrow**, the tooltip indicates that the number of relations that the arrow represents (see Figure 5). The second line of the tooltip displays aggregated information of the relation coloring attribute. In case of nominal data, the line shows the number of times each nominal value occurs. In case of ordinal data, the minimum, maximum, and average value is displayed.

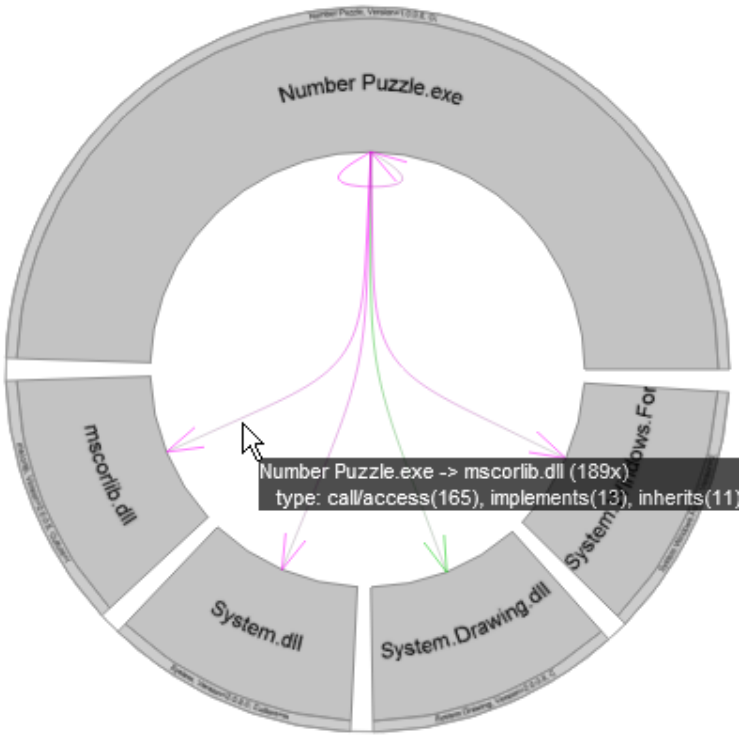


Figure 5: Aggregated information

3.3.2 Selection

When left-clicking on an element, the element and all of its descendants are selected. When holding control only the

element itself is selected, without its descendants. When holding shift the elements are added to the current selection. The selected elements are indicated using a black outline. To **clear the selection,** double-click the white background.

When a selection is made, only the relations that are incident to the selected elements are shown, the others are made gray. By changing the selection, also called the primary selection, the secondary selection is automatically updated. The secondary selection consists of elements that are connected to the primary selection by relations. Whether incoming or outgoing or relations are shown is determined by the "direction filter" found in the legend.

When **clicking on an arrow**, the two end elements are selected. You can select multiple arrows at once by holding the left-mouse button and then brushing the relations that should be selected.

3.3.3 Collapsing and expanding elements

As explained in Section 3.2.1, an important operation in SolidSX is the collapse and expansion of elements. A collapsed element does not show its children in the radial and tree view, so that the element takes up less screen space. All relations from or to any of the descendants of the collapsed element, go to the collapsed element instead.

Elements can be expanded or collapsed by double-clicking an element in the radial view or by brushing the node and pressing the **E** or **C** key, respectively. If there is no element under the mouse cursor, all elements in the selection are expanded or collapsed. Elements can be recursively expanded or collapsed by pressing **Shift-E** or **Shift-C**, respectively. In the tree view elements can be expanded or collapsed by clicking the little plus and minus signs left of the labels or by pressing the left and right arrow keys.

3.3.4 Hiding and disabling elements

A node can be hidden by pressing **H** when brushing an element. When hiding an element, the element takes up less space in the radial view, making more room for the others. Furthermore, relations from or to the hidden element are also hidden. Disabling an element has the same effect as hiding it, but disabling an element is more definite in the sense that it cannot be undone by selecting "Unhide all" (**Ctrl-U**). A hidden element that is not a leaf or collapsed still takes some screen space, the amount of which can be controlled from the settings window. Hidden leaf-elements do not take up any screen space.

3.3.5 Changing the element and relation coloring attributes

You can change the coloring attributes for both elements and relations from the legend. When brushing an attribute in the popup menu, the colors are immediately updated. This allows you to detect correlations among attributes.

3.3.6 Setting a relation filter

You can set two types of filters on relations: a filter on nominal data and a filter on ordinal data. When setting a filter, only relations that pass the filter are shown; the other relations are hidden.

To set a nominal filter, click Filter in the legend. A pop-up shows the available key=value pairs that can be used to filter the relations. These pairs are automatically determined by SolidSX when loading the

dataset. When brushing the filters, the result is immediately updated. To select multiple filters (each relation has to pass at least one of them), hold shift while selecting them. The popup disappears when releasing shift.

To set an ordinal filter, click Ordinal filter in the legend. A dockable pop-up appears. Select the key on which you want to set the filter. Using the two sliders you can set the minimum and maximum value that the attribute can attain. If a relation does not have the specified key, it does not pass the filter.

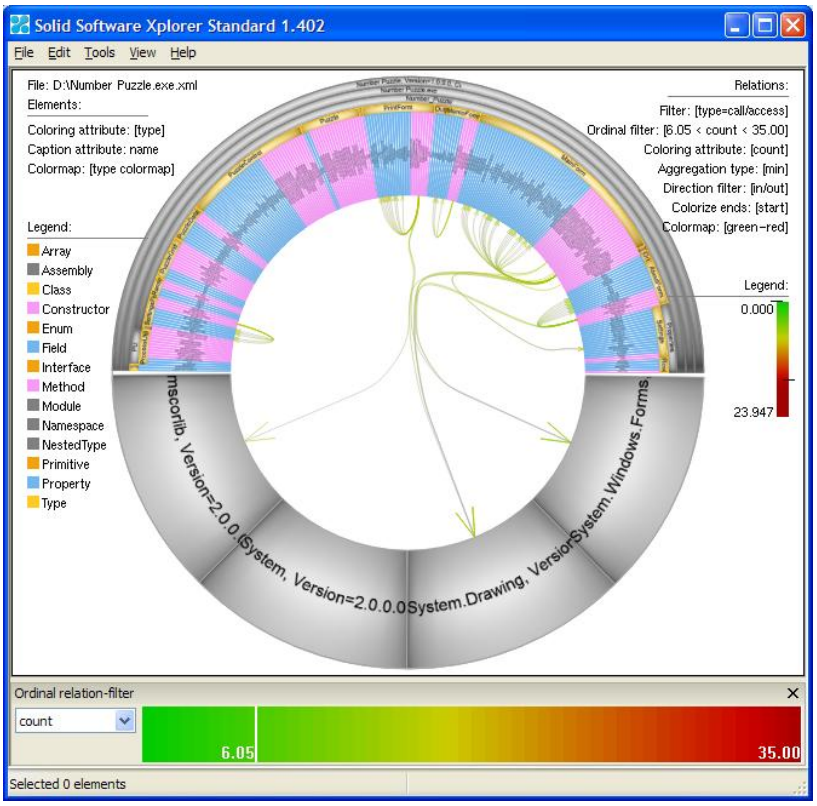


Figure 6: Putting an ordinal relation-filter on the count attribute.

3.3.7 Finding elements

It can be cumbersome to look for a specific code element by searching for it in the tree view or radial view. The code element might not even be visible in the current expansion state of the hierarchy. SolidSX allows you to find specific nodes by name or some other attribute. Choose “Tools -> Find” or press Ctrl-**F** to open the find dialog. First select the key on which to search and then type the search value that the attribute should contain. The status bar at the bottom of the main view indicates the number of selected nodes. You can use Python regular expressions by checking the option.

3.4 List of keyboard shortcuts and mouse commands

In the overview below, the "active element" denotes the brushed element (element under the mouse cursor), or when no element is brushed, the set of selected elements.

Radial view	
brush element	Show element tooltip
brush relation + Alt	Show relation tooltip
mouse left-click on element	Select element and its descendants
… + Control	Select element
… + Shift	Add element to selection
mouse left-click on relation(s)	Select the end elements of the relation(s)
mouse double left-click on white background	Clear selection
mouse left-click on color map legend box	Select all visible elements with this attribute
mouse double left-click on element	Expand element
mouse wheel on element	Increase width of element
mouse right-click on element/relation	Context menu
C	Collapse active element(s)
Shift + C	Collapse active element(s) and all descendants
E	Expand active element(s)
Shift + E	Expand active element(s) and all descendants
Ctrl +F	Find and select elements.
Ctrl + Z	Undo last collapse/expand command
←/→	Go to sibling of last selected element
↑ /↓	Go to parent/first child of last selected element
See the menu bar in SolidSX for more commands	

Tree view	
→	Expand element
shift + →	Expand element and all descendants
←	Collapse element
shift + ←	Collapse element and all descendants
↑/↓key	Go to parent/first child of element

3.5 Using the .NET importer

The .NET importer that comes with SolidSX extracts the information needed by SolidSX from executable and DLL files that are written in any .NET language, such as C#, Visual Basic, (managed) C++, and Silverlight. These .NET executables and DLLs are called assemblies. Because you do need to have the source code of the imported assembly, you can also inspect third-party assemblies, such as the .NET Core Library "mscorlib.dll" (however, note that some assemblies might be obfuscated).

In order to use the .NET importer, you need to have the Microsoft .NET Framework installed, which is probably installed already if you are a developer. The framework can be downloaded from http://microsoft.com/net.

To visualize your .NET project, select "File -> Import .NET assembly" and select the compiled executable or DLL file. The importer will output an intermediate SolidSX dataset to the folder "Program Files\SolidSX\datasets" and automatically open it. The "examples" folder located inside the SolidSX folder contains an example file "Number Puzzle.exe" to try out the .NET importer.

Datasets created using the .NET importer have the following element attributes (attributes denoted * are only available in SolidSX Standard edition):

- **name**: the name of the source code element. Note that some operators
- **type**: whether the element is a class, namespace, field, method, constructor, etc.
- **parametercount**: the number of parameters of a method
- **variablecount**: the number of local variables of a method
- **accessmodifier**: whether a class, field, or method is public, internal, etc.
- **callmodifier**: whether a method is virtual, abstract, etc.
- **codesize***: the size of the method body measured as the size of the compiled code in bytes
- **complexity***: the complexity of the method body measured as the number of branch statements
- **throwcount***: the number of throw statements inside a method
- **castcount***: the number of casts inside a method

Relations have the following attributes:

- **type:** the type of the relation. Can be "call/access", indicating a call relation between methods or field access, "inherits", for classes inheriting from other classes, and "implements", for classes implementing interfaces.
- **count:** the number of calls/field-accesses that the relation represents. Only for call/access relations.

3.6 The Solid Software Xplorer XML format

SolidSX is well-suited to visualize dependency data from any source and is not limited to software. You can visualize your own datasets using SolidSX by exporting your data to an XML file. Below is an example XML file demonstrating the format:

```
<?xml version="1.0" encoding="utf-8"?>
<dataset>
<nodes>
  <node nid="1">
    <attribute key="name" value="Root" />
  </node>
  <node nid="2">
    <attribute key="name" value="A" />
    <attribute key="weight" value="0.4" />
  </node>
  <node nid="3">
    <attribute key="name" value="B" />
  </node>
  <node nid="4">
    <attribute key="name" value="first child of A" />
  </node>
  <node nid="5">
    <attribute key="name" value="second child of A" />
    <attribute key="_icon" value="method" />
  </node>
</nodes>
<hierarchy name="default">
  <edge fromnid="1" tonid="2" />
  <edge fromnid="1" tonid="3" />
  <edge fromnid="2" tonid="4" />
  <edge fromnid="2" tonid="5" />
</hierarchy>
<edgeset name="default">
  <edge fromnid="3" tonid="4">
    <attribute key="type" value="call" />
  </edge>
  <edge fromnid="3" tonid="5">
    <attribute key="type" value="call" />
  </edge>
</edgeset>
</dataset>
```

SolidSX XML files contain three parts:

- <nodes>: list of all <node> XML elements.
- <hierarchy>: list of <edge> XML elements describing a hierarchy on the nodes. Multiple hierarchies per dataset are support. The active hierarchy can be changed from the legend.
- <edgeset>: list of <edge> elements describing adjacency relations. Currently, only one edge set per dataset is supported.

The three parts contain <node> and <edge> elements:

- Each node has an attribute `nid` that is a unique number identifying the node. Node id's should be positive and cannot be 0.
- Each edge element has attributes `fromnid` and `tonid` referencing nodes.

Both node and edges elements may contain <attribute> elements. An attribute element has an attribute `key` and `value`, assigning a key, value pair to the enclosing node or edge. Each node should contain at least the attribute "name".

There are some optional special attributes that affect the functioning of SolidSX, these attributes start with an underscore:

- Nodes can have an attribute "_icon" whose value is an image file that resides in the SolidSX subfolder "icons". The icon is shown in the tree view.
- Nodes can have an attribute "_primary" with values "True" and "False", indicating whether the element is from user code (primary), or is an element from an external library on which the primary code depends (secondary).
- Nodes can have an attribute "`_autodisablewhenexpanded`". Nodes having such an attribute (with another value than 0 or false), are automatically disabled when they are expanded. This is useful for elements that are not useful to have in the visualization at all times and that would often be hidden by the user anyway.

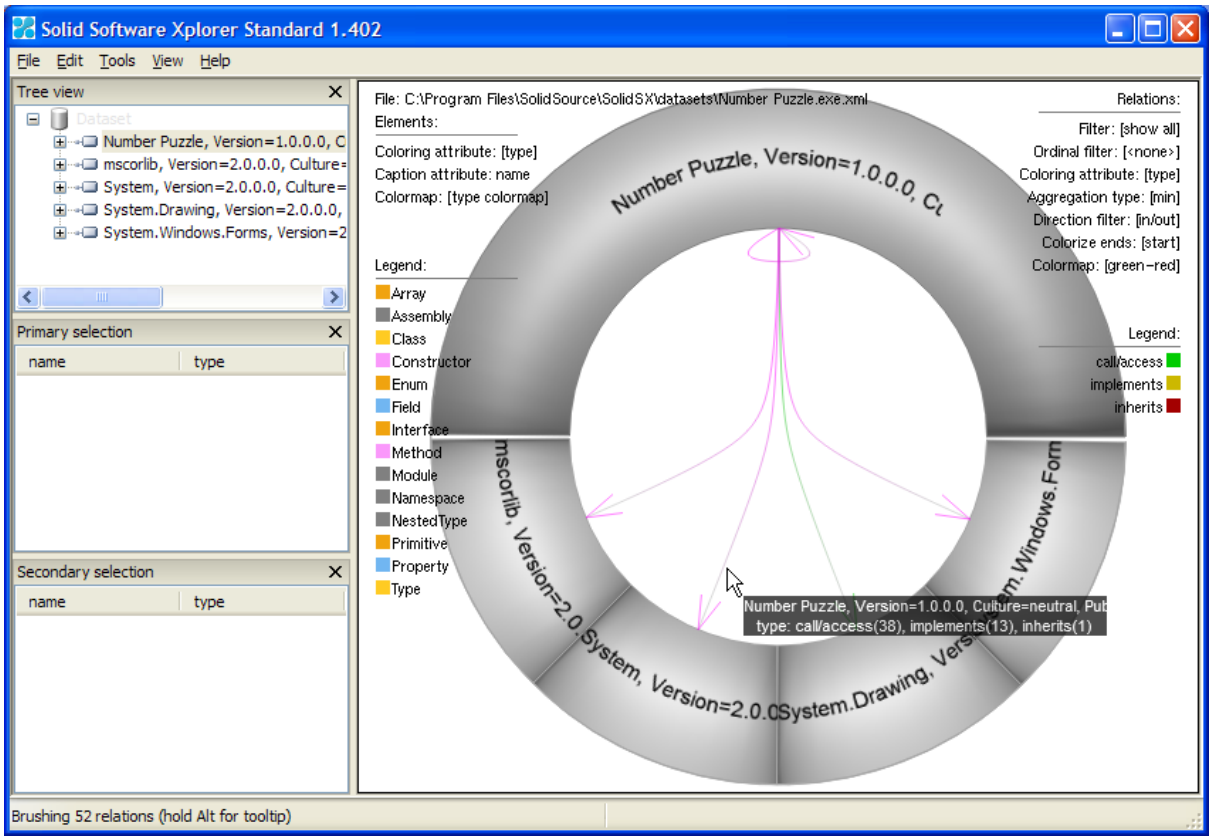
4 Tutorial

This section gives a short demonstration of some common operations in SolidSX. It is not meant as a typical use case.

In this tutorial we explore a simple example application that comes with SolidSX: an open-source Sudoku puzzle game written in C#. You can find the executable “Number Puzzle.exe” in the “examples” folder, located in the installation folder of SolidSX. The source code is also located there (zipped), but it is not needed for this tutorial.

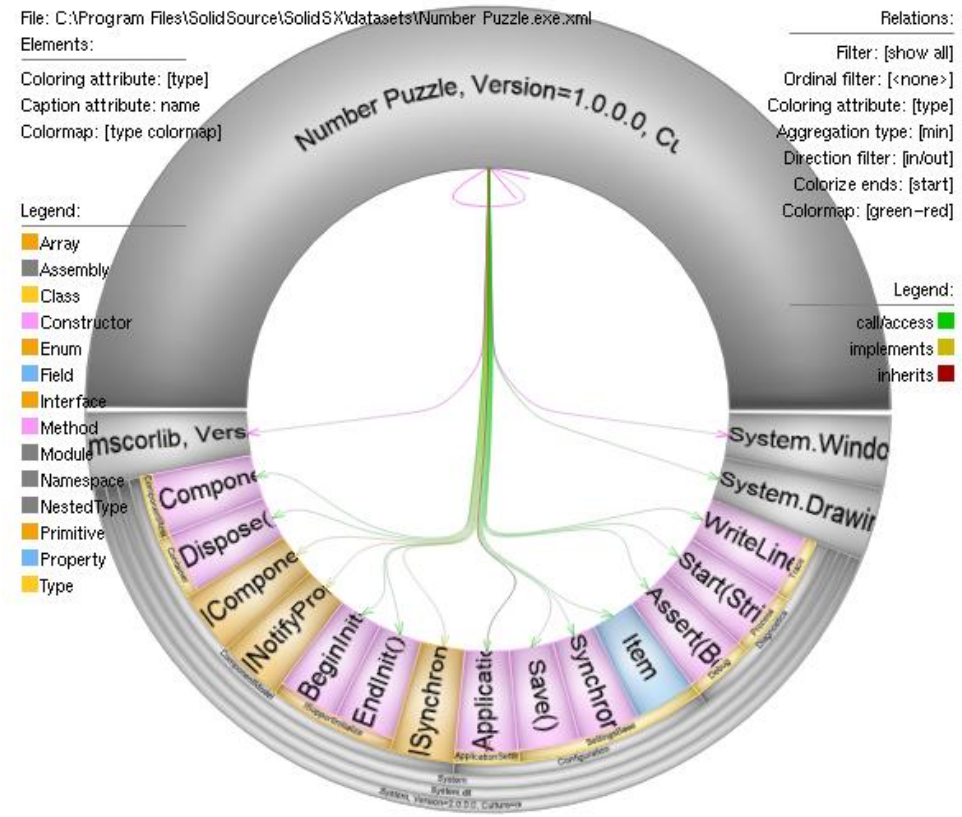
Please take the following steps:

1. Start SolidSX from the Windows Start menu.
2. Select File → Import .NET assembly
3. In the open file dialog, select the file “Number Puzzle.exe” in “Program Files\SolidSX\examples”.
4. SolidSX creates a dataset and opens it. You should see something like this:



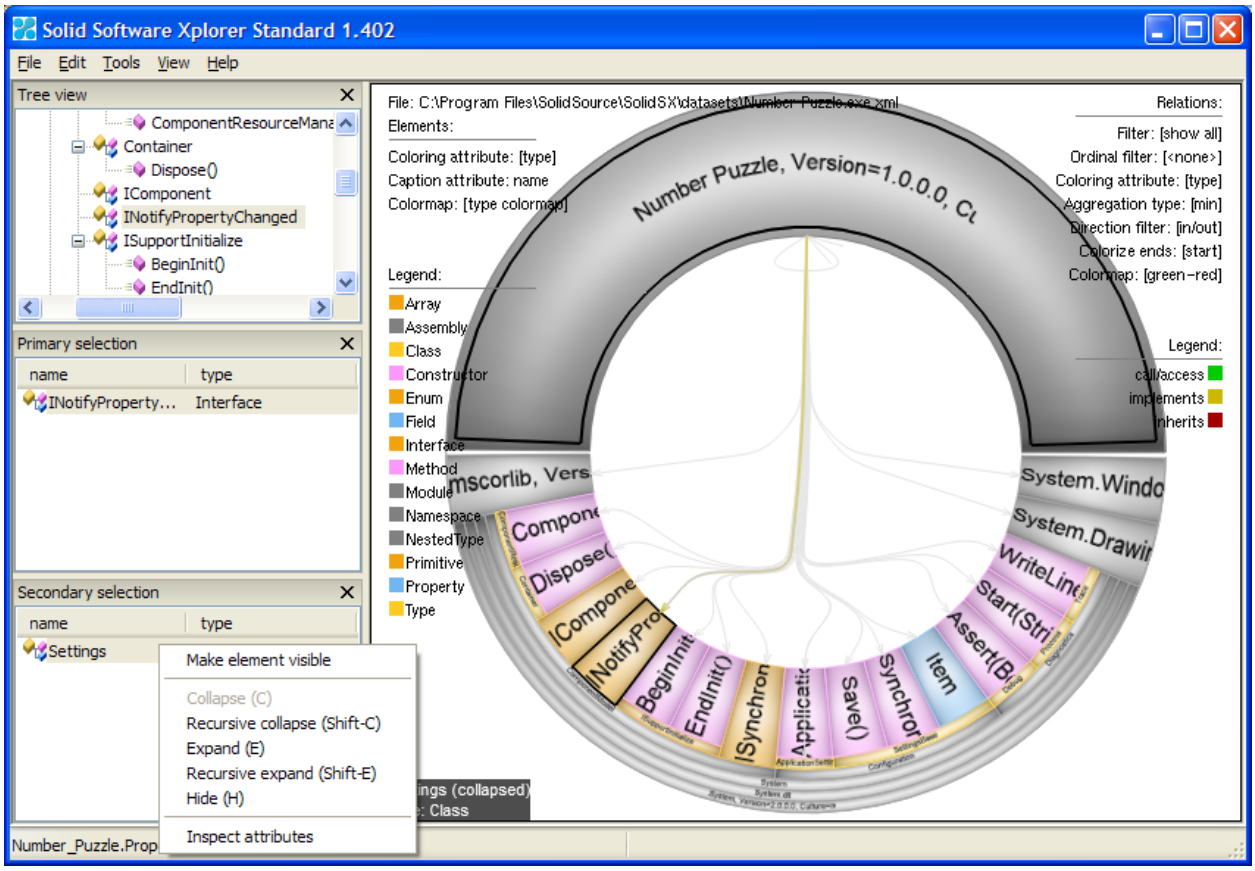
5. The radial view is divided into a top and bottom half. The top half contains elements from the user code, “Number Puzzle.exe” in this case, whereas the bottom half contains elements from external libraries. We see only 5 elements of the hierarchy, because they are (still) collapsed. If we brush an element, a tooltip appears that clearly states the name of the element and the value of the element coloring attribute, which is currently set to “type” (see the upper-left corner of the legend). By hovering over all 5 elements, we observe that they are all assemblies.

6. As we can see from the 5 arrows, Number Puzzle uses elements from the mscorlib, System, System.Drawing, and System.Windows.Forms assemblies. The self-loop indicates that Number Puzzle (obviously) depends on itself. Four of the five arrows are magenta colored, meaning that they contain at least two different values for the relation coloring attribute, which is currently also set to “type” (see the upper-right corner of the legend). If we brush the arrow (while holding ALT) going from “Number Puzzle” to “System”, the tooltip indicates that the arrow contains 38 relations with type=call/access, 13 relations having type=implements, and 1 relation having type=inherits. When brushing the green arrow, you can verify that the arrow only contains call/access relations.
7. We want to inspect the exact elements that “Number Puzzle” uses from “System”. We brush the element “System” and press **Shift+E** to fully expand the “System” element, until all the leaves of “System” are visible. We see the following:

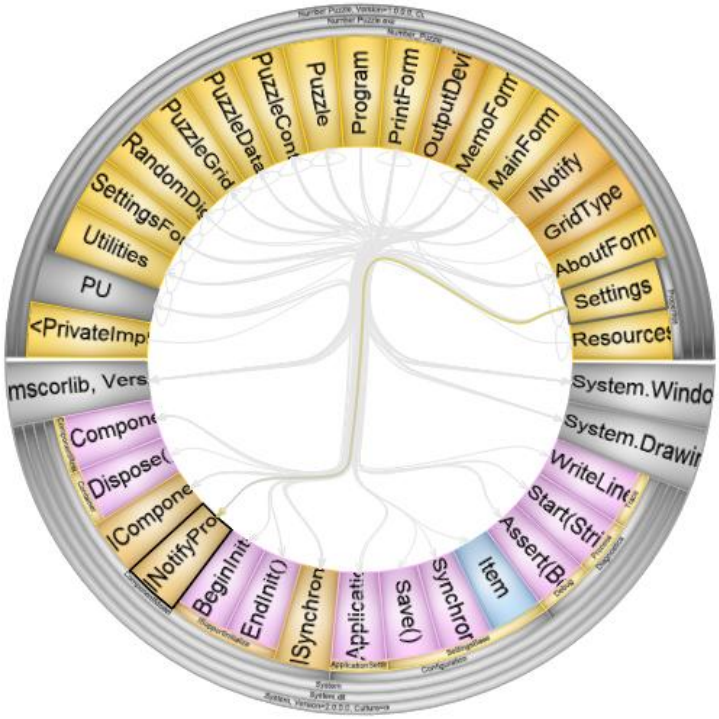


8. The colors of the elements that appeared are explained in the color map legend on the left-hand side. We see that the one magenta arrow that went from “Number Puzzle” to “System” is now split into several other arrows. Each arrow that appeared is green, yellow, or red, indicating a “call/access”, ‘implements”, or “inherits” relation respectively, as indicated by the color map legend on the right-hand side.
9. Suppose that we want to know which classes of Number Puzzle implement the “INotifyPropertyChanged” interface. We **left-click** the element “INotifyPropertyChanged”. The primary selection window is updated to reflect that the selection changed. The secondary selection is automatically updated by SolidSX and contains all elements that are connected to

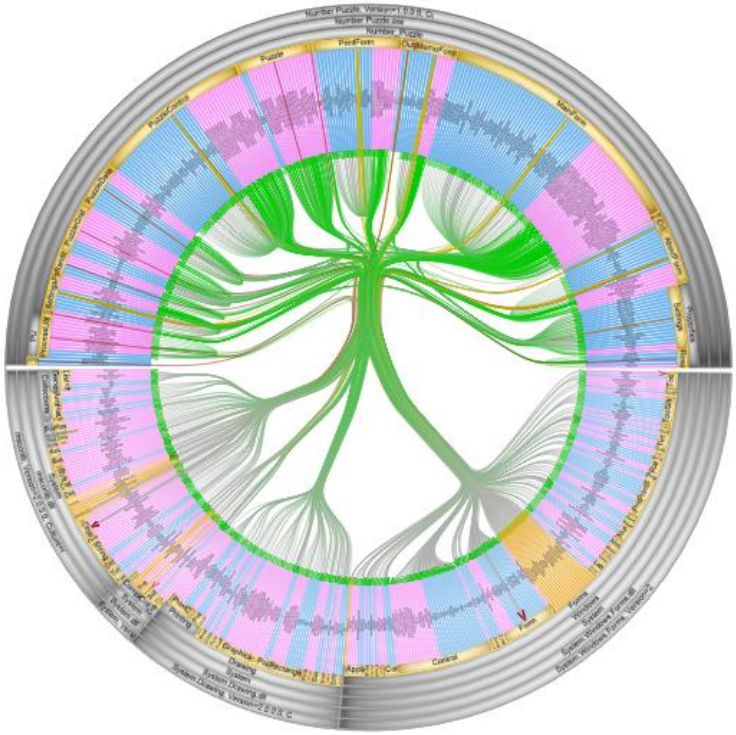
"INotifyPropertyChanged". We observe that only the class "Settings" is connected to the element:



The inset in "Number Puzzle" indicates that an element from the secondary selection is located inside it. To make the element visible in the radial view, we could expand "Number Puzzle" further (for example by pressing Shift+E or by using the tree view), or, we can right-click the "Settings" element in the secondary selection window and select "Make element visible" (see figure). Doing this, "Settings" is made visible by expanding all its ancestors, allowing you to see the context:



10. Finally, we expand all elements by pressing **Ctrl+E:**



11. This concludes the tutorial, but SolidSX contains many more features that you can discover by exploring the user interface.

5 Using the Visual Studio plug-in

SolidSX can operate in stand-alone mode as a normal Windows application, but can also be used as a plug-in for Microsoft Visual Studio. The advantage of using SolidSX as a plug-in is that you can more easily navigate between source code en visualization, even while you are coding. The plug-in has been verified to work with:

- Visual C# projects
- Visual Basic projects
- Visual C++ (managed) projects

For other .NET languages, the visualization works, but the code navigation options might not work.

The plug-in requires Microsoft Visual Studio 2008 Standard edition. Unfortunately, Microsoft does not allow plug-ins to work under the Express editions. When installing SolidSX, the Visual Studio plug-in is automatically installed if Microsoft Visual Studio 2008 is found. In the components panel of the setup you can see whether Visual Studio was found and whether the plug-in will be installed (see Figure 7).

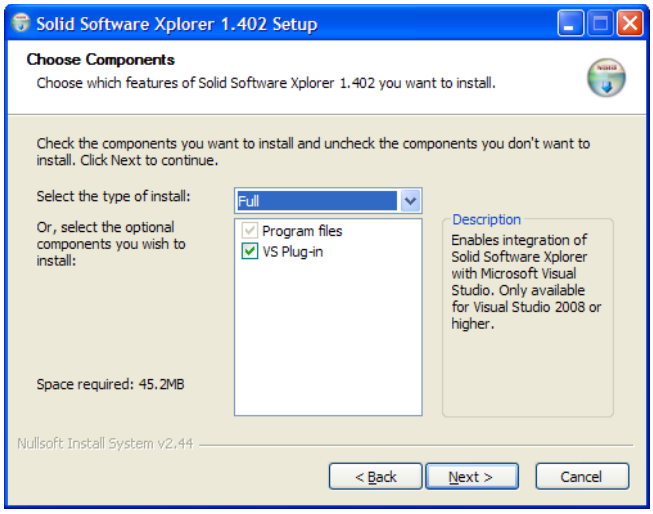


Figure 7: The setup only installs the plug-in if Visual Studio 2008 was found.

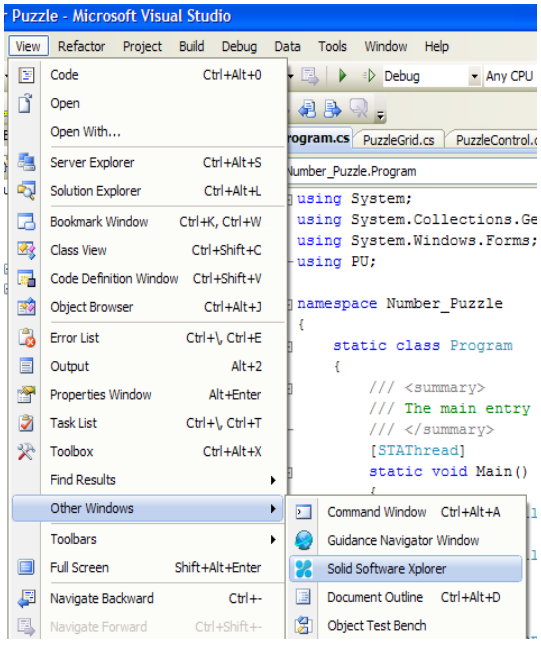


Figure 8: Opening SolidSX from Microsoft Visual Studio

When you start Microsoft Visual Studio after setup, there will be a new dockable window available that contains Solid Software Xplorer. To open the window, select View → Other windows → Solid Software Xplorer (see Figure 8).

Solid Software Xplorer behaves like any other tool window in Visual Studio: it can be docked, tabbed, hidden, etc (see Figure 9). The interface of SolidSX is almost the same as in stand-alone mode, with a few minor differences: the menu bar is not located at the top of the window anymore, but can now be

accessed through right-clicking the white background and selecting Menu. Furthermore, the status bar is always located underneath the radial view.

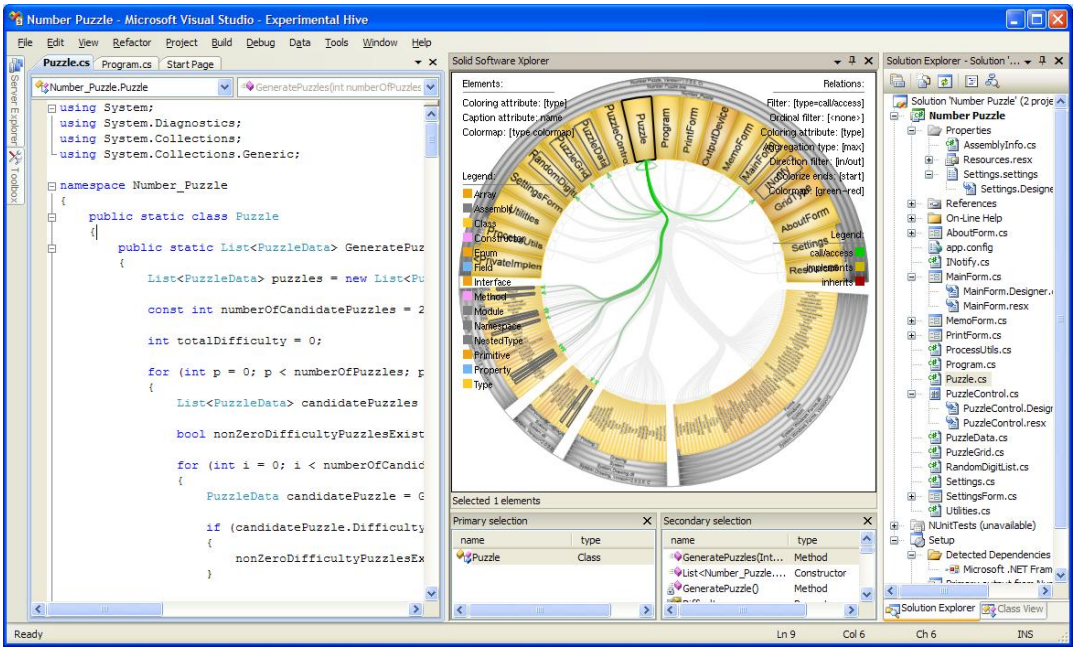


Figure 9: SolidSX integrates with Visual Studio like any other tool window.

As a plug-in, SolidSX allows you to navigate back and forth **between source code and the visualization**.

To go from a source code element to an element in the visualization, you simply put the text cursor somewhere inside or on the code element. For example, to select a method, set the cursor somewhere inside the method body. To select a field, set the cursor somewhere on the field name. The corresponding element is automatically added to the selection of SolidSX, as indicated by the black outline in the radial view. Like in the stand-alone version, the secondary selection is automatically computed, and contains the elements connected to the primary selection. The elements in the secondary selection are displayed using a gray outline. The primary and secondary selection windows are also updated accordingly. The secondary selection window can be used to quickly see the dependencies of the selected elements, for examples, the callers and callees for a certain method. Additionally, SolidSX highlights the visualization element on which the cursor is located by giving it a red outline.

Note that if the selected code element happens to be inside a collapsed ancestor in the visualization, the selection will be internal to the collapsed element. You could make the selection visible by right-clicking the element in the primary selection window and selecting “Make element visible”, for example.

To go from a visualization element to the corresponding source code element, right-click an element and select “Locate element in source code”. The source code element will be made visible in the text view and will receive a gray background. Right-clicking an element works in the radial view, as well as in the selection windows.

When you make changes to the source code, these changes are not reflected in the visualization until you recompile the project. The reason is that SolidSX does not analyze the source code directly, but analyzes the generated assembly instead.

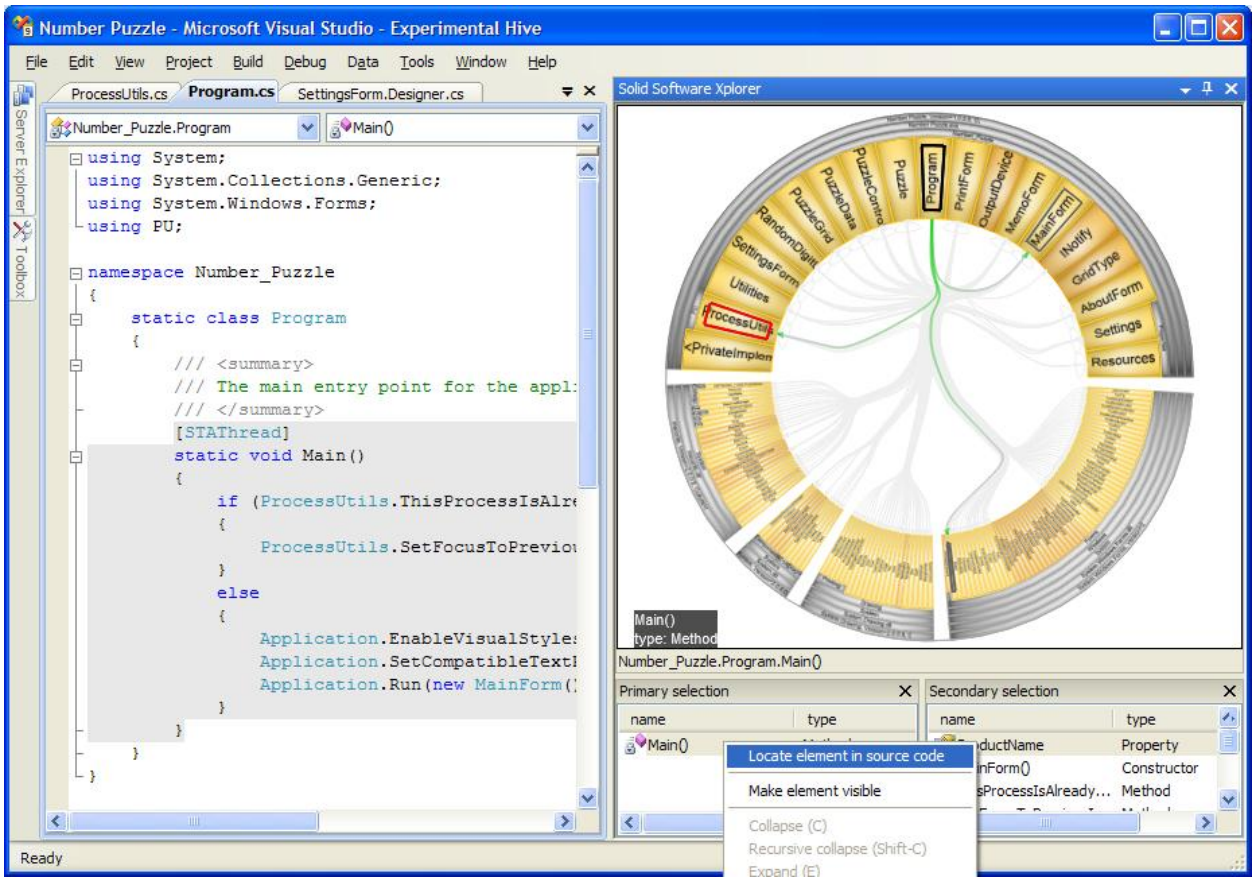


Figure 10: To go from visualization to source code, right-click an element and select "Locate element in source code". The corresponding code will get a gray background. To go from source code to the visualization, simply put the text cursor somewhere in a source code element.