CP Visualizer Format
February 15, 2010

Helmut Simonis

email: h.simonis@4c.ucc.ie
homepage: http://4c.ucc.ie/~hsimonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview
Outline

Contents

1 Introduction 2

2 Example Program 2

3 Search Tree Visualizer 5
  3.1 Example 5
  3.2 Schema 5
  3.3 Output 6

4 Constraint and Variable Visualizers 6
  4.1 Example 6
    4.1.1 Basic Types 6
    4.1.2 Structured Types 10
  4.2 Schema 10
  4.3 Output 10

5 Viz Program 11

6 VizTool Program 11

7 Conclusion 11

A Example Programs 16
  A.1 Bibsl 16
  A.2 Bin 17
  A.3 Car 18
  A.4 Costas 19
  A.5 Mix 20
  A.6 Nqueen 21
  A.7 Party 22
  A.8 Rooms 23
  A.9 Sendmore 24
  A.10 Sonet 25
  A.11 Sudoku 26
  A.12 Wave 27
Abstract

In this paper we describe a generic visualizer tool for finite domain constraint solvers, based on two data formats. The search tree visualizer allows to generate diagrams of the search tree of a finite domain constraint program. The constraint and variable visualizer displays information about the state of variables and constraints at different points of the computation. Both formats are using XML, and are described with their schemas in this document.

1 Introduction

Figure 1 shows the overall system architecture, for the ECLiPSe version of the tool. An annotated constraint program is run through the ECLiPSe system, and produces two log files, one for the search tree, the other for the constraints and variables. These log files are run through the Viz tool to generate visualization files in different formats. Most common are SVG based drawings, but treemaps, graphs and statistics can also be generated. The SVG files can be processed in different ways. They can be displayed in modern Web browsers, or can be edited with the inkscape SVG editor, for example to add annotations. inkscape can also be used in a batch process to convert the SVG drawings into other formats, in particular pdf for inclusion in LaTeX documents. Finally, the SVG files can be fed into the VizTool, an interactive SVG viewer linking multiple search tree and constraint visualizations.

To use the visualizer for another constraint system, the annotations in the source program need to be adjusted, and the log files must be generated during the execution of the tool.

Figure 2 shows the annotated ECLiPSe program to solve the SEND+MORE=MONEY puzzle, which highlights all required steps to interface the log generation with a constraint model. Before any constraint propagation is done, we create the visualization tool calling create_visualization, producing an opaque handle Handle. This handle gives us access to the visualization tool throughout the program, until we close the visualization tool.

This document describes the intermediate log formats for the search tree and constraint visualizer tools. These are XML files, each defined according to a schema detailed in the appendix. The document is structured as follows: In the next section we show an example annotated program (in ECLiPSe), to show how the logging steps interact with the constraint model. We then describe the details of the search tree visualizer in section 3. We give an example of the log format, discuss relevant points of the schema, and show some example output. In section 4 we repeat those steps for the constraint and variable visualizers. In section 5 we detail the interface of the viz tool, and in section 6 we describe the interactive viz tool.

2 Example Program

Figure 2 shows the annotated ECLiPSe program to solve the SEND+MORE=MONEY puzzle, which highlights all required steps to interface the log generation with a constraint model. Before any constraint propagation is done, we create the visualization tool calling create_visualization, producing an opaque handle Handle. This handle gives us access to the visualization tool throughout the program, until we close the visualization tool.
with close_visualization. We add a visualizer to the tool with the add_visualizer call, defining the type of the item to be visualized (here: vector) and the display format we want to use (here: expanded). When setting up constraints, we can explicitly call draw_visualizer to create a new snapshot of the model at this point. In this puzzle problem, we can name the variables with user-friendly labels, this is done with the name_variables call. Before starting the search, we call root to indicate that we are at the root of the search tree. The search routine uses a special version of the indomain assignment routine to automatically handle the updating of the logs at each search step. We describe the predicate tree_indomain below. Once a solution is found, we call the solution predicate to indicate that the search is finished and no further assignments will be done. At the end of the program, we close the visualizer with the close_visualization call, which will close the log files correctly, so that they can be processed with the viz tool.

Figure 2: Annotated SendMoreMoney Program

sendmory(L,Output,IgnoreFixed):-
L=[S,E,N,D,M,O,R,Y],
L :: 0..9,
create_visualization([output:Output,
                   ignore_fixed:IgnoreFixed,
                   width:8,
                   height:10],Handle),
add_visualizer(Handle,
              vector(L),
              [display:expanded]),
alldifferent(L),draw_visualization(Handle),
S #\= 0,draw_visualization(Handle),
M #\= 0,draw_visualization(Handle),
1000*S + 100*E + 10*N + D +
1000*M + 100*O + 10*R + E #=
10000*M + 1000*O + 100*N + 10*E + Y,
name_variables(Handle,L,
              ['S','E','N','D','M','O','R','Y'],
              Pairs),
root(Handle),
search(Pairs,1,input_order,
       tree_indomain(Handle,_),
       complete,[]),
solution(Handle),
close_visualization(Handle).

Figure 3 shows the internal operation of a visualizer aware indomain predicate. Depending on the value of the IgnoreFixed option, we may decide not to show any assignment steps which deal with already instantiated variables. The tree_indomain_generic routine gets the values in the domain into a list, possibly reordering the values depending on the assignment type Type and then iterates over the possible values in try_value. For each of the possible values V, it tries to set the variable X to this value. If this works, then it creates a TRY search node, indicating the assignment of the value V as the focus when drawing the visualization. If the assignment fails, it creates a FAIL tree node, and marks the failed assignment in the visualization, before forcing a fail to backtrack over the assignment. In the recursive call, we skip the first value, and test the remaining entries in the list of possible values.

The parameter Term of the indomain routine is not just a domain variable, but a term with multiple arguments. We extract the correct fields with multiple arg calls, using some parameters in the visualization structure. Ordinary ECLiPSe users should not have to understand the details of the tree_indomain_generic implementation, in most cases it is enough just to use a packaged search routine. We presented the details at this point to explain when the different nodes types of the search log are generated, and which state of the execution is captured.
tree_indomain_generic(Term,Handle,Handle,Type):-
    Handle = visualization(ignore_fixed:IgnoreFixed,
        var_arg:VarArg,
        name_arg:NameArg,
        focus_arg:FocusArg),
    arg(VarArg,Term,X),
    ((integer(X),IgnoreFixed = yes) ->
        true
    ;
        arg(NameArg,Term,Name),
        arg(FocusArg,Term,Focus),
        get_domain_as_list(X,L),
        get_domain_size(X,Size),
        reorganize_domain(X,L,Type,K),
        try_value(Handle,X,K,Name,Size,Focus)
    ).

try_value(Handle,X,[V|_],Name,Size,Focus):-
    ((X = V, true) ->
        try(Handle,Name,Size,V),
        focus_option(Focus,FocusOption),
        draw_visualization(Handle,FocusOption)
    ;
        failure(Handle,Name,Size,V),
        fail_option(Focus,V,FailOption),
        draw_visualization(Handle,FailOption),
        fail
    ).

try_value(Handle,X,[_|R],Name,Size,Focus):-
    try_value(Handle,X,R,Name,Size,Focus).
in the visualization logs.

There are four different search tree log node entries:

root There is a single root node with id 0, at the very top of the search tree. The corresponding visualizer snapshot is taken before the first assignment is attempted. It typically will show the effect of the constraint setup, e.g. domain reductions which have been performed while stating the constraints. The root node is generated by a call to the root predicate.

try A try node is generated whenever the assignment of a value to a variable succeeded. The visualizer snapshot shows the state after that assignment, and highlights the assigned variable.

fail A fail node is generated whenever the assignment of a value to a variable failed. The visualizer snapshot shows the state before that assignment, as this is the last fix-point that can be displayed. The visualizer also shows the failed assignment choice itself, but not any further constraint propagation which led to the failure. In particular, it does not show which domain was wiped out, or which constraint detected the failure.

succ Whenever a complete assignment which satisfies all constraints has been found, a succ node is generated. It marks the last choice as leading to a success. The visualization snapshot shows the state after the last assignment, i.e. there should be no variables left. The succ nodes are generated by a call to the solution predicate.

3 Search Tree Visualizer

In this section we describe the log format and the visualizer for the search tree. This can be used as a stand-alone tool to understand the search behavior of the program, without any variable or constraint visualizations.

3.1 Example

Figure 4 shows an example of the log format for the search tree. It is the search tree created when running the SEND+MORE=MONEY program above. The top-level element is tree, which contains a sequence of node elements of the four possible types. The first node must be a root node, the order of the following entries depends on the search. The log may contain zero, one or more succ nodes, the last node often is a succ node, but, as shown here, this is not always the case, depending on where the search is stopped. In this example we have explored all possible choice points, unfortunately this is often only possible for quite small problems.

Note how the tree element refers to the schema for the search tree log, given in the tree.xsd file. Using the schema is not required, but can help XML tools to validate a file based on the given structural information.

3.2 Schema

The tree.xsd schema is detailed in the appendix. Figure 5 shows a high-level structural reference. The tree element has a single version attribute (currently "1.0"), the succ and root node elements have a single id attribute, while try and fail elements have five attributes. They are

id This is a non-negative integer which serves as a reference to the node. Nodes should be numbered consecutively, starting with the root node with id 0.

parent This is a reference to the parent of the current node. This is a non-negative integer.

name This string holds the name of the selected variable. Often, this will be just an index in a list or vector rather than a specific name string. In our example file (figure 4) we have named the variables explicitly.

size This non-negative integer is the size of the domain of the selected variable before the assignment. This information can be helpful to understand the size of the unexplored part of the search tree.

value This is the integer value which is assigned to the variable at this node.

All attributes are required. A valid log file must have at least a single root node. Note that the log file is a flat structure, all nodes are children of the single tree element, i.e. the nodes are not nested according to the tree structure. The structure itself is maintained through the id and parent attributes.
3.3 Output

Figure 6 shows the resulting tree picture generated from the log file in figure 4. Try nodes are visualized as a node labelled with the variable name and a downward link labelled with the selected value. If multiple values for a variable are explored in a choice point, then the variable node is shared. Fail nodes are displayed as small red circles, while succ nodes are displayed as slightly larger, green circles. The root node is usually not displayed. Considering the links in upward direction shows the parent relation.

As the search tree grows in size, displaying the complete, expanded search tree becomes unwieldy. There are different options to compress failed sub-trees, or to only show some of the nodes in the tree. The latter option is also used to generate animations of the search progress, where a separate picture is drawn for each search node. Shown in sequence, they give an impression of the search progress.

4 Constraint and Variable Visualizers

In this section we describe the log format and visualizers for domain variables and constraints. While they can be used stand-alone, they normally refer to search tree nodes, so that a proper understanding of the behavior must integrate both logs.

4.1 Example

Figure 7 shows an excerpt of the log file generated from the example program in figure 2. An ellipsis (...) marks parts which have been removed. The top level element is visualization, which contains a non-empty set of visualizer elements followed by a set of state elements. The visualizer elements describe the different visualizers which should be shown in the visualization, the state elements describe execution states where snapshots for the variables and constraints are taken. Inside each state element, there are visualizer state elements, one for each visualizer. The states refer to the current search tree node with an attribute tree_node. This attribute has value -1 if the snapshot is taken outside the search, otherwise it is the id of the current node. The visualizer state element refers to its visualizer through its id attribute.

4.1.1 Basic Types

Inside each visualizer_state element we describe the state of its variables. This can be one of four types:
Figure 5: Search Tree Schema
Figure 6: Example Search Tree Output
Figure 7: Visualization Log Example

```xml
<visualization version="1.0" encoding="UTF-8" standalone="yes">
  <visualizer id="1" type="vector" display="expanded"
    x="0" y="0" width="8" height="10" min="0" max="9" />
  <state id="1" tree_node="1">
    <visualizer_state id="1">
      <dvar index="1" domain="0 .. 9"/>
      <dvar index="2" domain="0 .. 9"/>
      <dvar index="3" domain="0 .. 9"/>
      <dvar index="4" domain="0 .. 9"/>
      <dvar index="5" domain="0 .. 9"/>
      <dvar index="6" domain="0 .. 9"/>
      <dvar index="7" domain="0 .. 9"/>
      <dvar index="8" domain="0 .. 9"/>
    </visualizer_state>
  </state>
  ...
  <state id="5" tree_node="1">
    <visualizer_state id="1">
      <integer index="1" value="9"/>
      <dvar index="2" domain="4 .. 7"/>
      <dvar index="3" domain="5 .. 8"/>
      <dvar index="4" domain="2 .. 8"/>
      <integer index="5" value="1"/>
      <integer index="6" value="0"/>
      <dvar index="7" domain="2 .. 8"/>
      <dvar index="8" domain="2 .. 8"/>
      <focus index="1"/>
    </visualizer_state>
  </state>
  <state id="6" tree_node="2">
    <visualizer_state id="1">
      <integer index="1" value="9"/>
      <dvar index="2" domain="4 .. 7"/>
      <dvar index="3" domain="5 .. 8"/>
      <dvar index="4" domain="2 .. 8"/>
      <integer index="5" value="1"/>
      <integer index="6" value="0"/>
      <dvar index="7" domain="2 .. 8"/>
      <dvar index="8" domain="2 .. 8"/>
      <failed index="2" value="4"/>
    </visualizer_state>
  </state>
  ...
  <state id="14" tree_node="9">
    <visualizer_state id="1">
      <integer index="1" value="9"/>
      <integer index="2" value="5"/>
      <integer index="3" value="6"/>
      <integer index="4" value="7"/>
      <integer index="5" value="1"/>
      <integer index="6" value="0"/>
      <integer index="7" value="8"/>
      <integer index="8" value="2"/>
    </visualizer_state>
  </state>
  ...
</visualization>
```
**dvar** This element describes an unassigned variable, its position in the visualizer (attribute *index*), and its domain (attribute *domain*). If the visualizer is a one dimensional collection, the *index* is an integer, for two dimensional collections, it is a sequence of two integers, etc. The domain can be given as an interval \((1 \ldots 9)\) or as a white space separated list of integer values \(1 2 3 5 7\).

**integer** This element describes an assigned variable. The *index* attribute plays the same role as for *dvar* elements.

**svar** This element describes a set variable, given by lower and upper bounds on the set elements.

**sinteger** This is the basic type for a finite integer set.

**other** The element can be used by more complex global constraints to contain information which is not of one of the other basic types. An example would be atomic constraint types \(<, \leq, \neq\) used by some constraints.

**focus** This element describes a value assignment which should be highlighted in the visualizer. This is used in states which correspond to *try* search nodes. Only a single entry may be given.

**failed** This element describes a failed assignment. It marks the element which should be highlighted in the visualizer. This is used in states which correspond to *fail* search nodes. Only a single entry may be given, it can not be used in the same *visualizer_state* with a *focus* element.

### 4.1.2 Structured Types

If we want to visualize more complex global constraints, we can use additional structured types.

**argument** A number of (named or numbered) arguments can be given for a visualizer. Each argument may contain a single basic type, or a sequence of basic types, or more structured collection and tuple types. Arguments can only appear as direct children of *visualizer_state* elements, they can not be nested.

**collection** A collection is used to bind a sequence of other elements together. All elements inside a collection must have the same type, and can be other collections, tuples or basic types.

**tuple** A tuple provides fields for multiple elements which are accessed by name. They do not need to have the same type, and can be other collections, other tuples or basic types.

Figure 8 shows an excerpt of a log file for a cumulative constraint. The constraint has two arguments, the first is a sequence of tuples, the second a single limit value. Each tuple corresponds to a task, the fields can be accessed with named indices *start*, *dur* and *res*.

### 4.2 Schema

Figure 9 shows the high-level structure of the *visualization.xsd* schema, which is described in more detail in the appendix. The top level *visualization* element contains *visualizer* and *state* elements. Insides each *state* element we have a sequence of *visualizer_state* elements, which in turn contain elements for variables and constants as well as focus and failure information. The schema contains the structured elements for arguments, collections and tuples, but does not enforce all constraints on them, e.g. it does not know that all members of a collection should have same type.

### 4.3 Output

Figure 10 shows the output of the vector visualization in expanded form for a single state of execution. A vector is a one-dimensional sequence of variables, the expanded display shows each possible value in the domains as a separate field. Color coding is used to show which values have been assigned at this step (red), which values have been removed from the domain (blue) and which values remain in the domains (green). The display shown does not use the optional labels to save space in the display.
Figure 8: Example Log for Cumulative Constraint

5 Viz Program

The viz program is a command line tool which converts the log files into one or multiple drawings of some format. It is called with three arguments, all XML files. The first is a configuration file, which defines the tools that should be used to visualize the input data. The format of the configuration is detailed in the appendix, an example file is shown in figure 11.

More detailed documentation of the Viz program is given in its Javadoc description.

6 VizTool Program

The VizTool program is an interactive tool to display the visualizations along a time-line and to animate movement through the search. Figure 13 shows a snapshot. There is a time-line at the top, and two main panes, on the left for the search tree, on the right for the constraint visualization.

7 Conclusion

In this paper we have described the format for a generic CP visualizer, developed initially for an on-line constraint programming course using ECLiPSe. It uses XML based log files to collect information about the search, the
Figure 9: Visualization Log Schema

Figure 10: Example Visualization Output
variables and the constraints of the problem, and produces diagrams that can be used interactively, or processed for inclusion in documents and web sites.

The log formats are described using XML schemas, and Java tools to process and display the information are available under a Modzilla-type license for use with any constraint programming system.
Figure 12: Configuration Schema
Figure 13: VizTool: Car Sequencing
A Example Programs

The following pages show some snapshots from the visualization of different example programs written in ECLiPSe, generated by the Viz program as SVG output and converted into PDF documents through inkscape. We used the inkscape extension to run inkscape through a Makefile.

A.1 Bibd

Figure 14: Balanced Incomplete Block Design
A.2 Bin

Figure 15: Bin Packing Version of Party
A.3 Car

Figure 16: Car Sequencing with Sequence Constraints
A.4 Costas

Figure 17: Costas Array
A.5 Mix

Figure 18: Visualization Demo
A.6 Nqueen

Figure 19: N-Queen Puzzle
A.7 Party

Figure 20: Progressive Party Problem
A.8 Rooms

Figure 21: Rooms Puzzle
A.9 Sendmore

Figure 22: SEND+MORE=MONEY Puzzle
A.10  Sonet

Figure 23: Sonet Network Design
A.11 Sudoku

Figure 24: Sudoku

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>B</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
A.12 Wave

Figure 25: Routing and Wavelength Assignment
**Schema tree.xsd**

schema location:  `\sendmore\FULL\tree.xsd`
attribute form default:  `unqualified`
element form default:  `qualified`

Elements

**tree**

**element tree**

<table>
<thead>
<tr>
<th>diagram</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="tree_diagram.png" alt="tree diagram" /></td>
<td><img src="attributes.png" alt="attributes" /></td>
</tr>
</tbody>
</table>

**properties**

<table>
<thead>
<tr>
<th>content</th>
<th>complex</th>
</tr>
</thead>
</table>

**children**

- try
- succ
- root
- fail

**attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
<td>top level element describing a search tree</td>
</tr>
</tbody>
</table>

**source**

```xml
<xs:element name="tree">
  <xs:annotation>
    <xs:documentation>top level element describing a search tree</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="try">
          <xs:complexType>
            <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
            <xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
            <xs:attribute name="value" type="xs:integer" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="succ">
          <xs:complexType>
            <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="root">
        </xs:element>
        <xs:element name="fail">
        </xs:element>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
```xml
<x:element name="root">
  <x:complexType>
    <x:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
  </x:complexType>
</x:element>

<x:element name="fail">
  <x:complexType>
    <x:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
    <x:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
    <x:attribute name="name" type="xs:string" use="required"/>
    <x:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
    <x:attribute name="value" type="xs:integer" use="required"/>
  </x:complexType>
</x:element>
</x:choice>
</x:sequence>
<x:attribute name="version" type="xs:string" use="required"/>
</x:complexType>
</x:element>

attribute tree/@version
<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0, use required</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;version&quot; type=&quot;xs:string&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>

element tree/try

```

![Diagram](image)

<table>
<thead>
<tr>
<th>properties</th>
<th>isRef 0, content complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Name</td>
</tr>
<tr>
<td>id</td>
<td>xs:nonNegativeInteger</td>
</tr>
<tr>
<td>parent</td>
<td>xs:nonNegativeInteger</td>
</tr>
<tr>
<td>name</td>
<td>xs:string</td>
</tr>
<tr>
<td>size</td>
<td>xs:nonNegativeInteger</td>
</tr>
<tr>
<td>value</td>
<td>xs:integer</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:element name=&quot;try&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xs:complexType</a></td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;id&quot; type=&quot;xs:nonNegativeInteger&quot; use=&quot;required&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;parent&quot; type=&quot;xs:nonNegativeInteger&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>
```
attribute tree/try/@id

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td></td>
<td>use required</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;id&quot; type=&quot;xs:nonNegativeInteger&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>

attribute tree/try/@parent

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td></td>
<td>use required</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;parent&quot; type=&quot;xs:nonNegativeInteger&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>

attribute tree/try/@name

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td></td>
<td>use required</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;name&quot; type=&quot;xs:string&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>

attribute tree/try/@size

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td></td>
<td>use required</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;size&quot; type=&quot;xs:nonNegativeInteger&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>

attribute tree/try/@value

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td></td>
<td>use required</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;value&quot; type=&quot;xs:integer&quot; use=&quot;required&quot;/&gt;</td>
</tr>
</tbody>
</table>

element tree/succ

```
<xs:complexType>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="value" type="xs:integer" use="required"/>
</xs:complexType>
```
| properties | isRef 0 | complex |
| attributes | Name | Type | Use | Default | Fixed | annotation |
| source | <xs:element name="succ"> <xs:complexType> <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/> </xs:complexType> </xs:element> |

attribute tree/succ/@id

| properties | isRef 0 | required |
| type | xs:nonNegativeInteger |
| source | <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/> |

element tree/root

| properties | isRef 0 | complex |
| attributes | Name | Type | Use | Default | Fixed | annotation |
| source | <xs:element name="root"> <xs:complexType> <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/> </xs:complexType> </xs:element> |

attribute tree/root/@id

| properties | isRef 0 | required |
| type | xs:nonNegativeInteger |
| source | <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/> |
### Element tree/fail

<table>
<thead>
<tr>
<th>Diagram</th>
</tr>
</thead>
</table>
| ![Diagram](image)

<table>
<thead>
<tr>
<th>Properties</th>
<th>IsRef</th>
<th>Content</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>0</td>
<td>complex</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>xs:nonNegativeInteger</td>
<td>required</td>
</tr>
<tr>
<td>parent</td>
<td>xs:nonNegativeInteger</td>
<td>required</td>
</tr>
<tr>
<td>name</td>
<td>xs:string</td>
<td>required</td>
</tr>
<tr>
<td>size</td>
<td>xs:nonNegativeInteger</td>
<td>required</td>
</tr>
<tr>
<td>value</td>
<td>xs:integer</td>
<td>required</td>
</tr>
</tbody>
</table>

**Source**

```xml
<xs:element name="fail">
  <xs:complexType>
    <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="value" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
```

### Attribute tree/fail/@id

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>IsRef</th>
<th>Use</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Source**

```xml
<x:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
```

### Attribute tree/fail/@parent

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>IsRef</th>
<th>Use</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Source**

```xml
<x:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
```

### Attribute tree/fail/@name

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:string</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>IsRef</th>
<th>Use</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Source**

```xml
<x:attribute name="name" type="xs:string" use="required"/>
```
### attribute `tree/fail/@size`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
<tr>
<td></td>
<td>use: required</td>
</tr>
<tr>
<td>Source</td>
<td><code>&lt;xs:attribute name=&quot;size&quot; type=&quot;xs:nonNegativeInteger&quot; use=&quot;required&quot;/&gt;</code></td>
</tr>
</tbody>
</table>

### attribute `tree/fail/@value`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
<tr>
<td></td>
<td>use: required</td>
</tr>
<tr>
<td>Source</td>
<td><code>&lt;xs:attribute name=&quot;value&quot; type=&quot;xs:integer&quot; use=&quot;required&quot;/&gt;</code></td>
</tr>
</tbody>
</table>
### Schema `visualization.xsd`

- **Schema Location:** `\\courseware\documentation\visualization.xsd`
- **Attribute Form Default:** unqualified
- **Element Form Default:** qualified

#### Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Used By</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>xs:string</td>
<td>failed focus</td>
<td><code>&lt;xs:attribute name=&quot;group&quot; type=&quot;xs:string&quot;/&gt;</code></td>
</tr>
<tr>
<td>index</td>
<td>xs:string</td>
<td>argument collection dvar integer other sinteger state svar tuple visualization visualizer visualizer_state</td>
<td><code>&lt;xs:attribute name=&quot;index&quot; type=&quot;xs:string&quot;/&gt;</code></td>
</tr>
</tbody>
</table>

#### Elements

- `argument`
- `collection`
- `dvar`
- `failed`
- `focus`
- `integer`
- `other`
- `sinteger`
- `state`
- `svar`
- `tuple`
- `visualization`
- `visualizer`
- `visualizer_state`
**element argument**

**Diagram**

- **type**: extension of `collectable`
- **properties**: content complex
- **children**: `collection tuple dvar integer sinteger svar other`
- **used by**: element `visualizer_state`
- **attributes**: Name index Type Use Default Fixed annotation

**Source**

```xml
<xs:element name="argument">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="collectable">
        <xs:attribute ref="index" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```
**element collection**

```

diagram

<table>
<thead>
<tr>
<th>collection</th>
<th>collectable (extension)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>collection</td>
</tr>
<tr>
<td></td>
<td>dvar</td>
</tr>
<tr>
<td></td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>sinteger</td>
</tr>
<tr>
<td></td>
<td>svar</td>
</tr>
<tr>
<td></td>
<td>other</td>
</tr>
<tr>
<td></td>
<td>attributes</td>
</tr>
<tr>
<td></td>
<td>index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source</th>
</tr>
</thead>
</table>
| <xs:element name="collection">
|     <xs:complexType>
|         <xs:complexContent>
|             <xs:extension base="collectable">
|                 <xs:attribute ref="index" use="required"/>
|             </xs:extension>
|         </xs:complexContent>
|     </xs:complexType>
| </xs:element> |
```

**type** extension of collectable

**properties**

- content: complex

**children**

- collection
- tuple
- dvar
- integer
- sinteger
- svar
- other

**used by**

- complexType: collectable

**attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**source**

```
<xs:element name="collection">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="collectable">
                <xs:attribute ref="index" use="required"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
```

**element dvar**

```

diagram

<table>
<thead>
<tr>
<th>dvar</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>index</td>
</tr>
<tr>
<td></td>
<td>domain</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>content: complex</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>element: visualizer_state</td>
</tr>
<tr>
<td>complexTypes: collectable items</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>attributes</th>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>domain</td>
<td>xs:string</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source</th>
<th>&lt;xs:element name=&quot;dvar&quot;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="">xs:complexType</a></td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute ref=&quot;index&quot; use=&quot;required&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;domain&quot; type=&quot;xs:string&quot; use=&quot;required&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:complexType&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:element&gt;</td>
</tr>
</tbody>
</table>

attribute dvar/@domain

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0 use required</td>
</tr>
</tbody>
</table>

| source     | <xs:attribute name="domain" type="xs:string" use="required"/> |

element failed

<table>
<thead>
<tr>
<th>diagram</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>index</td>
</tr>
<tr>
<td></td>
<td>group</td>
</tr>
<tr>
<td></td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>properties</th>
<th>content complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>used by</td>
<td>element visualizer_state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attributes</th>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>group</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>xs:integer</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source</th>
<th>&lt;xs:element name=&quot;failed&quot;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="">xs:complexType</a></td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute ref=&quot;index&quot; use=&quot;required&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute ref=&quot;group&quot; use=&quot;required&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;value&quot; type=&quot;xs:integer&quot; use=&quot;required&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:complexType&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:element&gt;</td>
</tr>
</tbody>
</table>

attribute failed/@value

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0 use required</td>
</tr>
</tbody>
</table>

| source     | <xs:attribute name="value" type="xs:integer" use="required"/> |
**element focus**

Diagram:

```
  focus
     + attributes
     |   index
     |   group
     |   type
```

- **properties**
  - content: complex

- **used by**
  - element: `visualizer_state`

- **attributes**
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>group</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **source**

```xml
<x:element name="focus">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute ref="group" use="required"/>
    <xs:attribute name="type" type="xs:string" use="required"/>
  </xs:complexType>
</x:element>
```

**attribute focus/@type**

- **type**: xs:string

- **isRef**: 0

- **use**: required

- **source**

```xml
<x:attribute name="type" type="xs:string" use="required"/>
```

**element integer**

Diagram:

```
  integer
     + attributes
     |   index
     |   value
```

- **properties**
  - content: complex

- **used by**
  - element: `visualizer_state`
    - complexTypes: collectable items

- **attributes**
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>xs:integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **source**

```xml
<x:element name="integer">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="value" type="xs:integer" use="required"/>
  </xs:complexType>
</x:element>
```
attribute integer/@value

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>use</td>
<td>required</td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
</tbody>
</table>

source <xs:attribute name="value" type="xs:integer" use="required"/>

element other

<table>
<thead>
<tr>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
</tr>
<tr>
<td>value</td>
</tr>
</tbody>
</table>

| properties |
| content    |
| complex    |

| used by   |
| element   |
| visualizer_state |

| complexTypes |
| collectable_items |

<p>| attributes |
| Name |</p>
<table>
<thead>
<tr>
<th>index</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

source <xs:element name="other">
 <xs:complexType>
  <xs:attribute ref="index" use="required"/>
  <xs:attribute name="value" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>

attribute other/@value

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>use</td>
<td>required</td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
</tbody>
</table>

source <xs:attribute name="value" type="xs:string" use="required"/>

element sinteger

<table>
<thead>
<tr>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>sinteger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
</tr>
<tr>
<td>value</td>
</tr>
</tbody>
</table>

| properties |
| content    |
| complex    |

| used by   |
| element   |
| visualizer_state |

| complexTypes |
| collectable_items |
## attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```xml
<source>
<xs:element name="sinteger">
<xs:complexType>
  <xs:attribute ref="index" use="required"/>
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</source>
```

### attribute `sinteger/@value`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef: 0   use: required</td>
</tr>
</tbody>
</table>

```xml
<source>
<xs:attribute name="value" type="xs:string" use="required"/>
</source>
```

## element `state`

<table>
<thead>
<tr>
<th>properties</th>
<th>content complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>visualizer_state</td>
</tr>
</tbody>
</table>

### attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>xs:nonNegativeInteger</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```xml
source
<xs:element name="state">
<xs:annotation>
  <xs:documentation>
    describes the state of execution at some specific time point
  </xs:documentation>
</xs:annotation>
</xs:element>
```

### `visualizer_state`

- **annotation documentation**: describes the state of execution at some specific time point

```xml
<xs:element name="visualizer_state">  
  <xs:documentation>
    describes the state of execution at some specific time point
  </xs:documentation>
</xs:element>
```
point</xs:documentation>
</xs:annotation>
</xs:complexType>
<xs:sequence maxOccurs="unbounded">
<xs:element ref="visualizer_state"/>
</xs:sequence>
<xs:attribute name="id" type="xs:nonNegativeInteger" use="required">
<xs:annotation>
<xs:documentation>a sequential number defining the timepoint in execution</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="tree_node" type="xs:integer" use="required">
<xs:annotation>
<xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:element>

attribute state/@id

type  xs:nonNegativeInteger

properties  
isRef 0
use required

annotation  
documentation
a sequential number defining the timepoint in execution

source
<xs:attribute name="id" type="xs:nonNegativeInteger" use="required">
<xs:annotation>
<xs:documentation>a sequential number defining the timepoint in execution</xs:documentation>
</xs:annotation>
</xs:attribute>

attribute state/@tree_node

type  xs:integer

properties  
isRef 0
use required

annotation  
documentation
links the state to a node in the search tree, can be -1 if not inside search

source
<xs:attribute name="tree_node" type="xs:integer" use="required">
<xs:annotation>
<xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation>
</xs:annotation>
</xs:attribute>
**element svar**

![Diagram](image)

**properties**
- content: complex

**used by**
- element: `visualizer_state`
- complexTypes: `collectable_items`

<table>
<thead>
<tr>
<th>attributes</th>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>index</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>high</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**source**

```xml
<s:element name="svar">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="low" type="xs:string" use="required"/>
    <xs:attribute name="high" type="xs:string" use="required"/>
  </xs:complexType>
</s:element>
```

**attribute svar/@low**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>

**properties**
- isRef: 0
- use: required

**source**

```xml
<xs:attribute name="low" type="xs:string" use="required"/>
```

**attribute svar/@high**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>

**properties**
- isRef: 0
- use: required

**source**

```xml
<xs:attribute name="high" type="xs:string" use="required"/>
```
**element tuple**

<table>
<thead>
<tr>
<th>Type</th>
<th>extension of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>content complex</td>
</tr>
<tr>
<td>Children</td>
<td>dvar integer svar sinteger other tuple</td>
</tr>
<tr>
<td>Used by</td>
<td>complexTypes collectable items</td>
</tr>
<tr>
<td>Attributes</td>
<td>Name Type Use Default Fixed annotation</td>
</tr>
<tr>
<td></td>
<td>index required</td>
</tr>
</tbody>
</table>

```xml
<xs:element name="tuple">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="items">
                <xs:attribute ref="index" use="required"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
```
element visualization

```
<xs:element name="visualization">
  <xs:annotation>
    <xs:documentation>description of variables and constraint visualizers</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="visualizer" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>description of a constraint or a set of variables which will be visualized</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>id is referred to by visualizer_state</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="type" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>type of visualizer; must be supported on both sizes</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="display" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>how to display the visualizer</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="x" type="xs:integer" default="0">
            <xs:annotation>
              <xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
<xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation>
</xs:annotation>
</xs:attribute>
  <xs:attribute name="y" type="xs:integer" default="0"/>
</xs:annotation>
</xs:documentation>
</xs:attribute>
  <xs:attribute name="height" type="xs:integer" use="required"/>
</xs:annotation>
</xs:documentation>
</xs:attribute>
  <xs:attribute name="group" type="xs:string"/>
</xs:annotation>
</xs:documentation>
</xs:attribute>
  <xs:annotation>
    <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation>
  </xs:annotation>
</xs:complexType>
</xs:element>
  <xs:element name="state" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>describes the state of execution at some specific time point</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="visualizer_state"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:nonNegativeInteger" use="required">
        <xs:annotation>
          <xs:documentation>a sequential number defining the timepoint in execution</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="tree_node" type="xs:integer" use="required">
        <xs:annotation>
          <xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="version" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>currently "1.0"</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:complexType>
</xs:element>
attribute `visualization/@version`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0 use required</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation currently &quot;1.0&quot;</td>
</tr>
<tr>
<td>source</td>
<td><code>&lt;xs:attribute name=&quot;version&quot; type=&quot;xs:string&quot; use=&quot;required&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xs:annotation&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xs:documentation&gt;</code> currently &quot;1.0&quot;`</td>
</tr>
<tr>
<td></td>
<td><code>&lt;/xs:documentation&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/xs:attribute&gt;</code></td>
</tr>
</tbody>
</table>

element `visualization/visualizer`

| diagram | `id` | id is referred to by `visualizer_state` |
|         | `type` | type of visualizer; must be supported on both sizes |
|         | `display` | how to display the visualizer |
|         | `x` | optional explicit placement of visualizer on canvas |
|         | `y` |  |
|         | `width` |  |
|         | `height` |  |
|         | `group` | optional parameter; allows grouping of multiple constraints |
|         | `min` | expected minimal value of any of the domains |
|         | `max` |  |

| properties | isRef | 0 |
|            | minOcc | 1 |
|            | maxOcc | unbounded |
|            | content | complex |

| attributes | Name | Type | Use | Default | Fixed | annotation documentation |
|           | id   | xs:string | required |  |  | id is referred to by `visualizer_state` |
|           | type | xs:string | required |  |  | documentation type of |
### Document Content

**source**

```xml
<xs:element name="visualizer" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>
            description of a constraint or a set of variables which will be visualized
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="id" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>
                    id is referred to by visualizer_state
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="type" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>
                    type of visualizer; must be supported on both sizes
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="display" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>
                    how to display the visualizer
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="x" type="xs:integer" default="0">
            <xs:annotation>
                <xs:documentation>
                    optional explicit placement of visualizer on canvas
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="y" type="xs:integer" default="0">
            <xs:annotation>
                <xs:documentation>
                    optional explicit placement of visualizer on canvas
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="width" type="xs:integer" use="required">
            <xs:annotation>
                <xs:documentation>
                    documentation
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="height" type="xs:integer" use="required">
            <xs:annotation>
                <xs:documentation>
                    documentation
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="group" type="xs:string">\n            <xs:annotation>
                <xs:documentation>
                    optional parameter, allows grouping of multiple constraints
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="min" type="xs:integer">
            <xs:annotation>
                <xs:documentation>
                    expected minimal value of any of the domains
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="max" type="xs:integer">
            <xs:annotation>
                <xs:documentation>
                    expected minimal value of any of the domains
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
</xs:element>
```
attribute `visualization/visualizer/@id`  
<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>
| properties| isRef 0  
use required |
| annotation| documentation  
id is referred to by visualizer_state |
| source    | `<xs:attribute name="id" type="xs:string" use="required">  
<xs:annotation>  
<xs:documentation>id is referred to by visualizer_state</xs:documentation>  
</xs:annotation>  
</xs:attribute>` |

attribute `visualization/visualizer/@type`  
<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>
| properties| isRef 0  
use required |
| annotation| documentation  
type of visualizer; must be supported on both sizes |
| source    | `<xs:attribute name="type" type="xs:string" use="required">  
<xs:annotation>  
<xs:documentation>type of visualizer; must be supported on both sizes</xs:documentation>  
</xs:annotation>  
</xs:attribute>` |

attribute `visualization/visualizer/@display`  
<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>
| properties| isRef 0  
use required |
| annotation| documentation  
how to display the visualizer |
| source    | `<xs:attribute name="display" type="xs:string" use="required">  
<xs:annotation>  
<xs:documentation>how to display the visualizer</xs:documentation>  
</xs:annotation>  
</xs:attribute>` |
### attribute `visualization/visualizer/@x`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
<tr>
<td></td>
<td>default: 0</td>
</tr>
</tbody>
</table>

**Annotation**
- **documentation**: optional explicit placement of visualizer on canvas

**Source**
```xml
<xs:attribute name="x" type="xs:integer" default="0">
  <xs:annotation>
    <xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

### attribute `visualization/visualizer/@y`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
<tr>
<td></td>
<td>default: 0</td>
</tr>
</tbody>
</table>

**Source**
```xml
<xs:attribute name="y" type="xs:integer" default="0"/>
```

### attribute `visualization/visualizer/@width`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
<tr>
<td></td>
<td>use: required</td>
</tr>
</tbody>
</table>

**Source**
```xml
<xs:attribute name="width" type="xs:integer" use="required"/>
```

### attribute `visualization/visualizer/@height`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
<tr>
<td></td>
<td>use: required</td>
</tr>
</tbody>
</table>

**Source**
```xml
<xs:attribute name="height" type="xs:integer" use="required"/>
```

### attribute `visualization/visualizer/@group`

<table>
<thead>
<tr>
<th>Type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>isRef: 0</td>
</tr>
</tbody>
</table>

**Annotation**
- **documentation**: optional parameter, allows grouping of multiple constraints

**Source**
```xml
<xs:attribute name="group" type="xs:string">
  <xs:annotation>
    <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation>
  </xs:annotation>
</xs:attribute>
```
attribute visualization/visualizer/@min

type xs:integer

properties
- isRef: 0
- default: 0

annotation documentation expected minimal value of any of the domains

source
<xs:attribute name="min" type="xs:integer" default="0">
    <xs:documentation>
        expected minimal value of any of the domains
    </xs:documentation>
</xs:attribute>

attribute visualization/visualizer/@max

type xs:integer

properties
- isRef: 0
- default: 0

source
<xs:attribute name="max" type="xs:integer" default="0"/>

element visualization/state

diagram

- state
  - id
    - a sequential number defining the timepoint in execution
  - tree_node
    - links the state to a node in the search tree, can be -1 if not inside search

properties
- isRef: 0
- minOcc: 1
- maxOcc: unbounded
- content: complex

children
- visualizer_state

attributes
- id
  - Type: xs:nonNegativeInteger
  - Use: required
- tree_node
  - Type: xs:integer
  - Use: required

annotation documentation describes the state of execution at some specific time point
<xs:element name="state" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>describes the state of execution at some specific time point</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element ref="visualizer_state"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:nonNegativeInteger" use="required">
      <xs:annotation>
        <xs:documentation>a sequential number defining the timepoint in execution</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="tree_node" type="xs:integer" use="required">
      <xs:annotation>
        <xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>

attribute visualization/state/@id

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef: 0, use: required</td>
</tr>
<tr>
<td>annotation</td>
<td>a sequential number defining the timepoint in execution</td>
</tr>
</tbody>
</table>

attribute visualization/state/@tree_node

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef: 0, use: required</td>
</tr>
<tr>
<td>annotation</td>
<td>links the state to a node in the search tree, can be -1 if not inside search</td>
</tr>
</tbody>
</table>

source
### element visualizer

**Diagram**

- **id**
  - id is referred to by `visualizer_state`

- **type**
  - type of visualizer; must be supported on both sizes

- **display**
  - how to display the visualizer
    - **x**
      - optional explicit placement of visualizer on canvas
    - **y**
    - **width**
    - **height**
    - **group**
      - optional parameter, allows grouping of multiple constraints
      - **min**
        - expected minimal value of any of the domains
      - **max**

**Description**

- description of a constraint or a set of variables which will be visualized

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>xs:string</td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>xs:integer</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>xs:integer</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>xs:integer</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>xs:integer</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>group</td>
<td>xs:string</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
grouping of multiple constraints

documentation expected minimal value of any of the domains

### Annotation Documentation

Description of a constraint or a set of variables which will be visualized.

```xml
<xs:element name="visualizer">
    <xs:annotation>
        <xs:documentation>description of a constraint or a set of variables which will be visualized</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="id" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>id is referred to by visualizer_state</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="type" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>type of visualizer; must be supported on both sizes</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="display" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>how to display the visualizer</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="x" type="xs:integer" default="0">
            <xs:annotation>
                <xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="y" type="xs:integer" default="0"/>
        <xs:attribute name="width" type="xs:integer" default="0"/>
        <xs:attribute name="height" type="xs:integer" default="0"/>
        <xs:attribute name="group" type="xs:string">
            <xs:annotation>
                <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="min" type="xs:integer" default="0">
            <xs:annotation>
                <xs:documentation>expected minimal value of any of the domains</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="max" type="xs:integer" default="0"/>
    </xs:complexType>
</xs:element>
```
**attribute visualizer/@id**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>
| properties | isRef 0  
use required |
| annotation | documentation |
| source | `<xs:attribute name="id" type="xs:string" use="required">`  
`<xs:annotation>`  
`<xs:documentation>`id is referred to by visualizer_state` `<xs:annotation>`  
`</xs:attribute>` |

**attribute visualizer/@type**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>
| properties | isRef 0  
use required |
| annotation | documentation |
| source | `<xs:attribute name="type" type="xs:string" use="required">`  
`<xs:annotation>`  
`<xs:documentation>`type of visualizer; must be supported on both sizes` `<xs:annotation>`  
`</xs:attribute>` |

**attribute visualizer/@display**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
</table>
| properties | isRef 0  
use required |
| annotation | documentation |
| source | `<xs:attribute name="display" type="xs:string" use="required">`  
`<xs:annotation>`  
`<xs:documentation>`how to display the visualizer` `<xs:annotation>`  
`</xs:attribute>` |

**attribute visualizer/@x**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
</table>
| properties | isRef 0  
default 0 |
| annotation | documentation |
| source | `<xs:attribute name="x" type="xs:integer" default="0">`  
`<xs:annotation>`  
`<xs:documentation>`optional explicit placement of visualizer on canvas` `<xs:annotation>`  
`</xs:attribute>` |
### attribute `visualizer/@y`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
</table>
| properties | isRef: 0  
default: 0 |

**source**

```xml
<xs:attribute name="y" type="xs:integer" default="0"/>
```

### attribute `visualizer/@width`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
</table>
| properties | isRef: 0  
default: 0 |

**source**

```xml
<xs:attribute name="width" type="xs:integer" default="0"/>
```

### attribute `visualizer/@height`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
</table>
| properties | isRef: 0  
default: 0 |

**source**

```xml
<xs:attribute name="height" type="xs:integer" default="0"/>
```

### attribute `visualizer/@group`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef: 0</td>
</tr>
</tbody>
</table>

**annotation**

```xml
<xs:annotation>
  <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation>
</xs:annotation>
```

**source**

```xml
<xs:attribute name="group" type="xs:string">
  <xs:annotation>
    <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

### attribute `visualizer/@min`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
</table>
| properties | isRef: 0  
default: 0 |

**annotation**

```xml
<xs:annotation>
  <xs:documentation>expected minimal value of any of the domains</xs:documentation>
</xs:annotation>
```

**source**

```xml
<xs:attribute name="min" type="xs:integer" default="0">
  <xs:annotation>
    <xs:documentation>expected minimal value of any of the domains</xs:documentation>
  </xs:annotation>
</xs:attribute>
```
attribute visulizer/@max

<table>
<thead>
<tr>
<th>type</th>
<th>xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td></td>
<td>default 0</td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;max&quot; type=&quot;xs:integer&quot; default=&quot;0&quot;/&gt;</td>
</tr>
</tbody>
</table>

element visulizer_state

Properties

- content: complex

Children

- argument
dvar
integer
sinteger
svar
other
failed
focus

Used by

- elements: visualization/state state

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Use</th>
<th>Default</th>
<th>Fixed</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>xs:nonNegativeInteger</td>
<td>required</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source

<xs:element name="visulizer_state">
  <xs:annotation>
    <xs:documentation>Comment describing your root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:choice>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="argument"/>
      </xs:sequence>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="dvar"/>
        <xs:element ref="integer"/>
        <xs:element ref="sinteger"/>
        <xs:element ref="svar"/>
        <xs:element ref="other"/>
        <xs:element ref="failed"/>
        <xs:element ref="focus"/>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
attribute `visualizer_state/@id`

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>use</td>
<td>required</td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
</tbody>
</table>

complexType `collectable`

```
<xs:complexType name="collectable">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="collection"/>
      <xs:element ref="tuple"/>
      <xs:element ref="dvar"/>
      <xs:element ref="integer"/>
      <xs:element ref="sinteger"/>
      <xs:element ref="svar"/>
      <xs:element ref="other"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```
complexType **items**

```xml
<xs:complexType name="items">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="dvar"/>
      <xs:element ref="integer"/>
      <xs:element ref="svar"/>
      <xs:element ref="sinteger"/>
      <xs:element ref="other"/>
      <xs:element ref="tuple"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

XML Schema documentation generated by [XMLSpy Schema Editor](http://www.altova.com/xmlspy)
Schema *configuration.xsd*

**schema location:** configuration.xsd  
**attribute form default:** unqualified  
**element form default:** qualified

**Elements**

*configuration*

**element configuration**

<table>
<thead>
<tr>
<th>properties</th>
<th>content</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>tool</td>
<td></td>
</tr>
<tr>
<td>attributes</td>
<td>version</td>
<td>xs:string</td>
</tr>
<tr>
<td></td>
<td>directory</td>
<td>xs:string</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
<td>Comment describes the configuration for the viz program</td>
</tr>
</tbody>
</table>

**source**

```xml
<xs:element name="configuration">
  <xs:annotation>
    <xs:documentation>Comment describes the configuration for the viz program</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="tool">
        <xs:complexType>
          <xs:attribute name="show" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>tree or viz</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="type" type="xs:string" use="optional" default="layout">
            <xs:annotation>
              <xs:documentation>layout, distribution, treemap</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
<xs:attribute name="display" type="xs:string" use="optional" default="compact">
  <xs:annotation>
    <xs:documentation>compact or expanded</xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute name="repeat" type="xs:string" use="optional" default="final">
  <xs:annotation>
    <xs:documentation>all, final , i or -i</xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute name="width" type="xs:nonNegativeInteger" use="optional" default="500">
  <xs:annotation>
    <xs:documentation>width of SVG canvas in screen pixels</xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute name="height" type="xs:nonNegativeInteger" use="optional" default="500">
  <xs:annotation>
    <xs:documentation>height of SVG canvas in screen pixels</xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute name="fileroot" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>root name of output files</xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute name="version" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>currently 1.0</xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute name="directory" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>name for directory where output is placed</xs:documentation>
  </xs:annotation>
</xs:attribute>

<table>
<thead>
<tr>
<th>attribute</th>
<th>type</th>
<th>properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration//@version</td>
<td>xs:string</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>xs:string</td>
<td>required</td>
</tr>
<tr>
<td>directory</td>
<td>xs:string</td>
<td>required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source</th>
</tr>
</thead>
</table>
| <xs:attribute name="version" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>currently 1.0</xs:documentation>
  </xs:annotation>
</xs:attribute> |
attribute configuration/@directory

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
</tr>
</tbody>
</table>

```xml
<xs:attribute name="directory" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>name for directory where output is placed</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

element configuration/tool

```xml
<xs:element name="tool" type="xs:string">
  <xs:annotation>
    <xs:documentation>tree or viz</xs:documentation>
    <xs:documentation>layout, distribution, treemap</xs:documentation>
    <xs:documentation>layout, distribution, treemap</xs:documentation>
    <xs:documentation>compact or expanded</xs:documentation>
    <xs:documentation>all, final, i or -i</xs:documentation>
    <xs:documentation>width of SVG canvas in screen pixels</xs:documentation>
    <xs:documentation>height of SVG canvas in screen pixels</xs:documentation>
    <xs:documentation>root name of output files</xs:documentation>
  </xs:annotation>
</xs:element>
```

<table>
<thead>
<tr>
<th>properties</th>
<th>isRef 0</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Name</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>show</td>
<td>xs:string</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>xs:string</td>
</tr>
<tr>
<td></td>
<td>display</td>
<td>xs:string</td>
</tr>
<tr>
<td></td>
<td>repeat</td>
<td>xs:string</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>xs:nonNegativeInteger</td>
</tr>
<tr>
<td>attribute</td>
<td>configuration/tool/@show</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>xs:string</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>isRef 0</td>
<td></td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>&lt;xs:attribute name=&quot;show&quot; type=&quot;xs:string&quot; use=&quot;required&quot;&gt;</td>
<td></td>
</tr>
</tbody>
</table>

source</xs:element>

<xs:element name="tool">
  <xs:complexType>
    <xs:attribute name="show" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>tree or viz</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:string" use="optional" default="layout">
      <xs:annotation>
        <xs:documentation>layout, distribution, treemap</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="display" type="xs:string" use="optional" default="compact">
      <xs:annotation>
        <xs:documentation>compact or expanded</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="repeat" type="xs:string" use="optional" default="final">
      <xs:annotation>
        <xs:documentation>all, final, i or -i</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="width" type="xs:nonNegativeInteger" use="optional" default="500">
      <xs:annotation>
        <xs:documentation>width of SVG canvas in screen pixels</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="height" type="xs:nonNegativeInteger" use="optional" default="500">
      <xs:annotation>
        <xs:documentation>height of SVG canvas in screen pixels</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="fileroot" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>root name of output files</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
attribute configuration/tool/@type

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>layout</td>
</tr>
<tr>
<td>use</td>
<td>optional</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
</tr>
<tr>
<td></td>
<td>layout, distribution, treemap</td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;type&quot; type=&quot;xs:string&quot; use=&quot;optional&quot; default=&quot;layout&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xs:annotation</a></td>
</tr>
<tr>
<td></td>
<td><a href="">xs:documentation</a>layout, distribution, treemap&lt;/xs:documentation&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:annotation&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:attribute&gt;</td>
</tr>
</tbody>
</table>

attribute configuration/tool/@display

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>compact</td>
</tr>
<tr>
<td>use</td>
<td>optional</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
</tr>
<tr>
<td></td>
<td>compact or expanded</td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;display&quot; type=&quot;xs:string&quot; use=&quot;optional&quot; default=&quot;compact&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xs:annotation</a></td>
</tr>
<tr>
<td></td>
<td><a href="">xs:documentation</a>compact or expanded&lt;/xs:documentation&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:annotation&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:attribute&gt;</td>
</tr>
</tbody>
</table>

attribute configuration/tool/@repeat

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>final</td>
</tr>
<tr>
<td>use</td>
<td>optional</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
</tr>
<tr>
<td></td>
<td>all, final , i or -i</td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;xs:attribute name=&quot;repeat&quot; type=&quot;xs:string&quot; use=&quot;optional&quot; default=&quot;final&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xs:annotation</a></td>
</tr>
<tr>
<td></td>
<td><a href="">xs:documentation</a>all, final , i or -i&lt;/xs:documentation&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:annotation&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:attribute&gt;</td>
</tr>
</tbody>
</table>

attribute configuration/tool/@width

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>isRef</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>500</td>
</tr>
<tr>
<td>use</td>
<td>optional</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>source</td>
<td>width of SVG canvas in screen pixels</td>
</tr>
</tbody>
</table>

```xml
<xs:attribute name="width" type="xs:nonNegativeInteger" use="optional" default="500">
  <xs:annotation>
    <xs:documentation>width of SVG canvas in screen pixels</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

**attribute configuration/tool/@height**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:nonNegativeInteger</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0 default 500 use optional</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation height of SVG canvas in screen pixels</td>
</tr>
</tbody>
</table>

```xml
<xs:attribute name="height" type="xs:nonNegativeInteger" use="optional" default="500">
  <xs:annotation>
    <xs:documentation>height of SVG canvas in screen pixels</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

**attribute configuration/tool/@fileroot**

<table>
<thead>
<tr>
<th>type</th>
<th>xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>isRef 0 use required</td>
</tr>
<tr>
<td>annotation</td>
<td>documentation root name of output files</td>
</tr>
</tbody>
</table>

```xml
<xs:attribute name="fileroot" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>root name of output files</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

XML Schema documentation generated by [XMLSpy Schema Editor](http://www.altova.com/xmlspy)