

ORCAN GUI

How to create and modify the user
interface of a component

Horst Hadler (LGDV), Flaviu Jurma (CGL,FHG-IISB)

ORCAN GUI

“GUI development is inflexible because the visual appearance is tightly coupled with the control’s behavior”

...

“All a button needs to know is to execute an action when checked”...

“the rest is visual appearance”

[CUJ 4/2005 – ‘Revolutionize your GUI’]

ORCAN GUI - Motivation

- Main intention: the component developer should be relieved from GUI matters
 - Separation of input variables (called properties in ORCAN) and their user interface
 - Extra file specifies layout, domain and behavior of input variables
 - All aspects of a variable's GUI (i.e. domain, layout, behavior) can be changed at runtime
 - Variables are automatically managed by GUI

Example: integer variable

1. In the component: add your variable to the PropertyMap (ORCAN's reflective container)

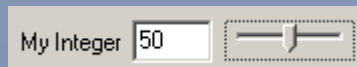
```
GetProperties().AddProperty( "Parameter:MyInteger", mMyInt );
```

adds the member variable mMyInt to the component's PropertyMap; it can now be accessed by the GUI

2. In a text file associated with the component: add the variable's description

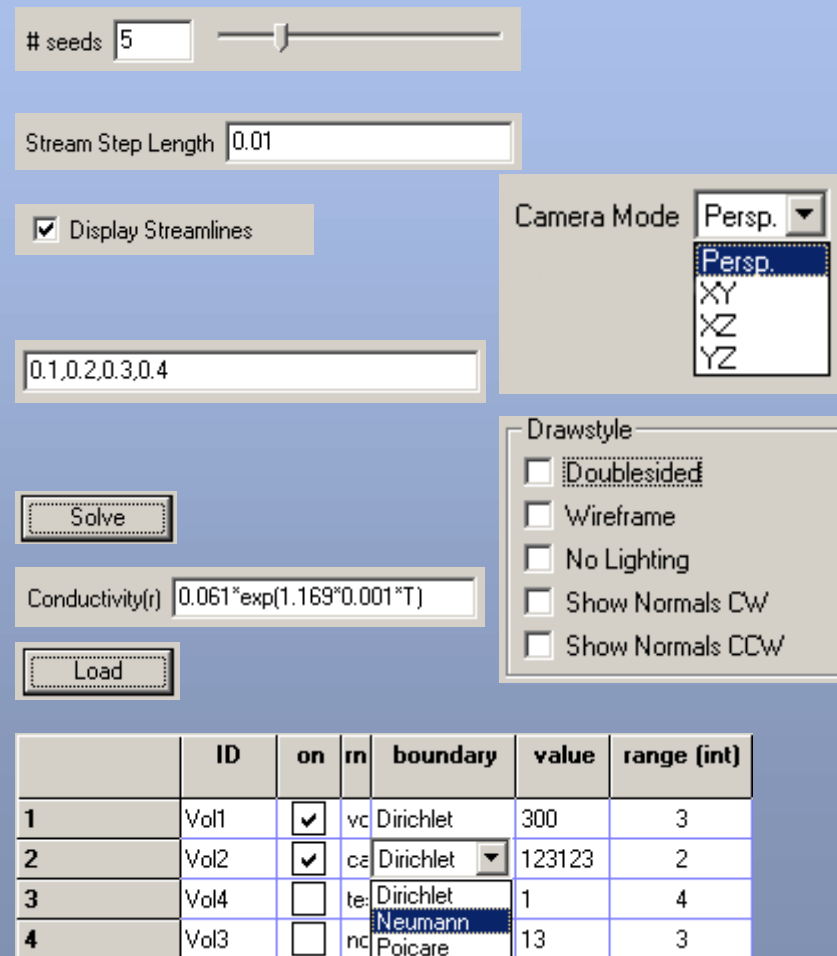
```
<resource>
  <name>Parameter:MyInteger</name>
  <integer> <start>1</start> <end>100</end> <step>1</step> </integer>
  <description>
    <guihint> slider </guihint>
    <label> My Integer</label>
  </description>
</resource>
```

3. Run the program



GUI Elements

- Integer: input represents an integer value
- Real: input represents a floating point value
- Bool: input represents a boolean value
- Enum: input is selection of one value from a group of values
- String: input represents a text value
- Bitgroup: input is a combination of powers of 2 ($1-2^{32}$)
- Action: input triggers a function call
- Formula: input represents a formula + evaluation and validation
- File: input represents a filename (+ dialogs)
- Table: input represents custom data in table form



	ID	on	rn	boundary	value	range (int)
1	Vol1	<input checked="" type="checkbox"/>	vc	Dirichlet	300	3
2	Vol2	<input checked="" type="checkbox"/>	ca	Dirichlet	123123	2
3	Vol4	<input type="checkbox"/>	te	Dirichlet	1	4
4	Vol3	<input type="checkbox"/>	nc	Neumann	13	3
				Poicare		

Format of GUI file

- XML (eXtensible Markup Language)
- ```
<?xml version="1.0" encoding="iso-8859-1"?>
<resources>
```

```
<resource>
 <name>NAME</name> ...
</resource>
...
```

GUI element  
description  
(for each variable)

```
<layout>
 <elements>
 <element><name>NAME</name></elements>
 ...
 </elements>
</layout>
```

Layout/Grouping  
of the elements

```
<rules>
 ...
</rules>
```

Behavior of GUI  
(connections between elements)

```
</resources>
```

# GUI file <resource>

- Specify name, domain and 'style' of an element/variable

```
<resource>
```

```
<name> Property:NAME </name>
```

variable's name

```
<integer>
```

```
<start>50</start> <end>1000</end> <step>50</step>
```

type and domain  
of GUI element

```
</integer>
```

```
<description>
```

```
<guihint> slider, textinput </guihint>
```

```
<label font="arial">
```

```
Its an int
```

```
</label>
```

```
<layout> framed </layout>
```

```
<tooltip> the tooltip </tooltip>
```

```
<border> 5 </border>
```

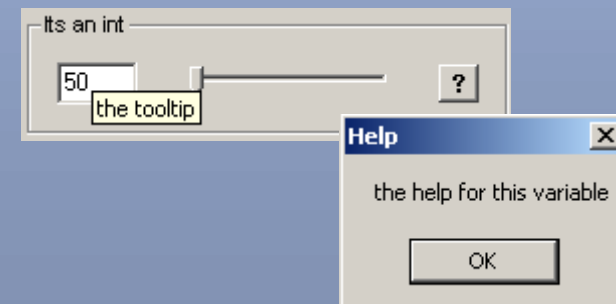
```
<help> this will be the help </help>
```

```
<orientation> horizontal </orientation>
```

```
</description>
```

layout and style  
of GUI element

```
</resource>
```



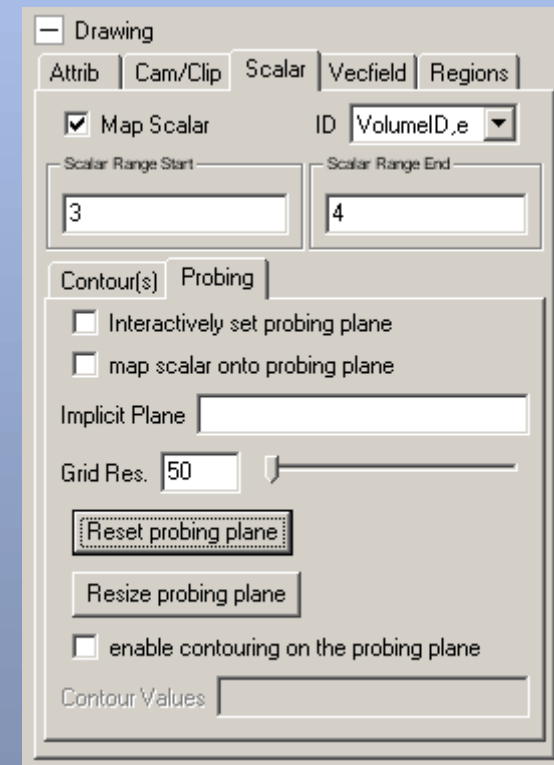
# GUI file <layout>

- Specifies arrangement of GUI elements and groups of GUI elements in rows and columns
- If omitted, all elements will be shown  
If given, only the elements specified will be shown
- Basic structure:
 

```

<layout>
 <elements>
 <resource> Parameter:NAME </resource>
 ... resources
 <group>
 ... elements and groups
 </group>
 </elements>
</layout>

```
- Groups can have a label and a layout
- Instead of group 'notebook' may be used; sub groups are 'notebookpages'





# GUI file <rules>

- Rules describe dependencies between elements
- Rules operate on the 'state' of the GUI elements: value, enable, visible, readonly, domain, layout, notify, trigger, defer

- Basic structure – by example:

```
<rules>
```

```
<action> Parameter:MyInteger.value=12 </action>
<action> Parameter:Int2.enable=false </action>
```

default values

```
<rule>
 <if>
 <condition> Parameter:MyInteger.value == 10 </condition>
 <action> Parameter:Int2.enable = true </action>
 </if>
 <elseif>
 <condition> Parameter:MyInteger.value LT 10 </condition>
 <action> Parameter:Int2.value = 0 </action>
 </elseif>
 <else>
 <action> Parameter:Int2.enable = false </action>
 </else>
</rule>
```

rule  
'if, elseif, else' block

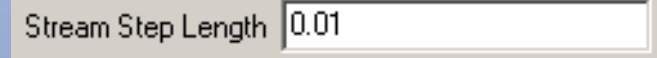
```
... other rules
```

```
</rules>
```

# <integer>/<real>/<bool> elements

- integer/real resource description:

```
<real>
 <start>0</start>
 <end>10</end>
 <step>0.001</step>
</real>
```

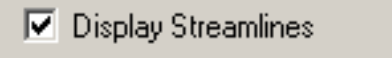
A screenshot of a GUI input field. It has a light gray background and a thin border. The text "Stream Step Length" is on the left, and a white text box on the right contains the value "0.01".

Stream Step Length 0.01

GUI hints: "slider"\*, "textinput", "spinbutton"

- bool resource description:

```
<bool>
 <label> Display Streamlines </label>
</bool>
```

A screenshot of a GUI checkbox. It consists of a small square box with a checkmark inside, followed by the text "Display Streamlines".

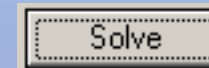
Display Streamlines

GUI hints: "checkboxbox"\*

# <action> element

- Calls a method in the component
- resource description (type and domain):

```
<action>
 <label> Solve </label>
</action>
```



- GUI hints: "button"\*
- Special AddProperty method for actions:

```
AddPropertyCallback("Parameter:TheName",
 this, &ClassName::MethodName);
```

- When button is pressed, method is called:

```
bool ClassName::MethodName(std::string const& name)
 { /* do s.th. */ return true; }
```

# <enum>/<bitgroup> elements

For selection from list and 'bitwise ors'

- resource description: name/value pairs

```
<enum>
```

```
 <entry> <name>Persp</name> <value>P_Def</value> </entry>
```

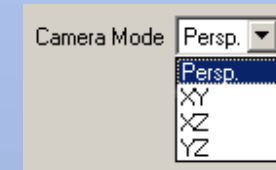
```
 <entry> <name>XY</name> <value>P_XY</value> </entry>
```

```
 <entry> <name>XZ</name> <value>P_XZ</value> </entry>
```

```
 ...
```

```
</enum>
```

- GUI hints: "combobox"\*, "radiobox"



- resource description: name/2<sup>x</sup> pairs

```
<bitgroup>
```

```
 <entry> <name>Doublesided</name> <value>1</value> </entry>
```

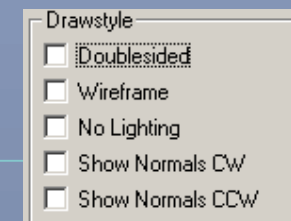
```
 <entry> <name>Wireframe</name> <value>2</value> </entry>
```

```
 <entry> <name>No Lighting</name> <value>4</value> </entry>
```

```
 ...
```

```
</bitgroup>
```

- GUI hints: "checkboxgroup"\*

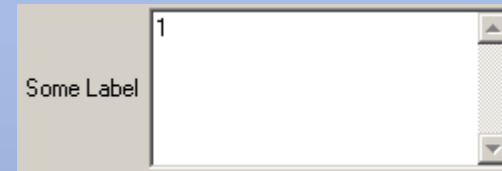


# <string> element

- For single and multi-line text
- resource description:

```
<string>
 <label> Some Label </label>
</string>
```

- label is optional
- GUI hints: "textinput"



- The layout tag in the elements description can have a format attribute. It describes the size of the textinput field. A single value of for example 10 describes a text-input with "10" columns. A value of "10x5" describes a text-input with 5 lines each having 10 columns.

```
<description>
 <layout format="10x5"> notframed </layout>
 ...
</description>
```

# <file> element

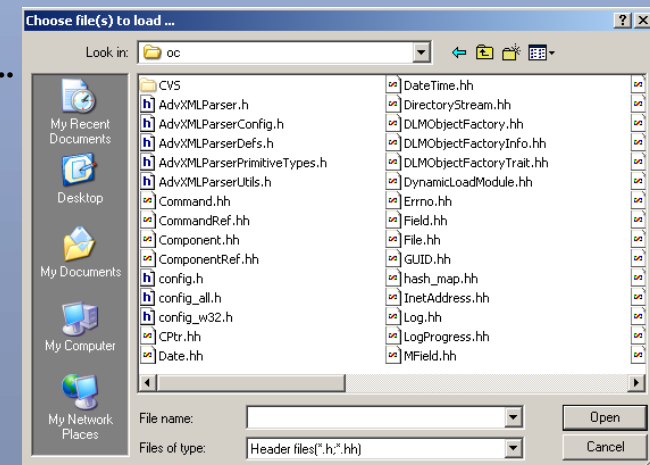
- Get filename(s)
- resource description:

```

<file>
 <type>File</type>
 <mode>Load</mode>
 <filter>
 Header files(*.h;*.hh) | *.h;*.hh |
 Source files(*.c;*.cpp) | *.c;*.cpp
 </filter>
 <dialogtitle>Choose file(s) to load ...
</dialogtitle>
</file>

```

- GUI hints: "textinput", "button"
- <type>: "File"\*, "Directory"
- <mode>: "Load"\*, "Save"; Load supports multi-selection
- <filter>: info/wildcard pairs



# <table> element

- Edit tabular data ( à oc::Table)
- resource description:

```

<table>
 <header>
 <cols>
 <col readonly="yes">
 <name>ID</name>
 <type>choice</type>
 <domain>Vol1,Vol2,Vol3,Vol4</domain>
 <layout width="4"></layout>
 </col>
 ... other col elements
 </cols>
 </header>
</table>

```

- <name> the column label
- <type> "string"\*, "bool", "real", "integer", "choice"
- <domain>
  - real: width,precision
  - integer: min,max
  - choice: comma separated list
  - string, bool: -
- <layout> width, align

	ID	on	rn	boundary	value	range (int)
1	Vol1	<input checked="" type="checkbox"/>	vc	Dirichlet	300	3
2	Vol2	<input checked="" type="checkbox"/>	ca	Dirichlet	123123	2
3	Vol4	<input type="checkbox"/>	te	Dirichlet	1	4
4	Vol3	<input type="checkbox"/>	nc	Neumann Poisone	13	3

- The layout tag in the element's description defines the size of the table: the number of rows and columns.

```

<description>
 <layout format="4x6">
 framed
 </layout>
 ...
</description>

```

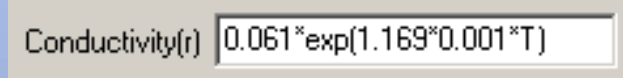
# <formula> element

- evaluate and validate input formulas ( à ocst::Formula)

- resource description:

```
<formula> </formula>
```

- GUI hint: "textinput"

A screenshot of a GUI text input field. The label "Conductivity(r)" is on the left, and the input field contains the formula "0.061\*exp(1.169\*0.001\*T)".

Conductivity(r) 0.061\*exp(1.169\*0.001\*T)

- usage:

```
ocst::Formula mFormula;
```

```
mFormula.SetVariables("T");
```

```
AddProperty("Parameter:MyF", mFormula.GetString());
```

```
AddPropertyCallback("Validator:MyF", &mFormula,
&ocst::Formula::IsValid);
```

- if validation fails, i.e. 'validator' returns false, a message box appears



# Validator callbacks

---

- Mechanism for variable validation
- Usage: Add parameter and validator callback

```
AddProperty("Parameter:MyParam" , mMyParam) ;
```

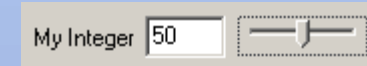
```
AddPropertyCallback("Validator:MyParam" ,
ClassPtr, &Class::ValidationMethod) ;
```

- Signature of validator callback is:

```
bool methodname(std::string const &val)
```

# Example: change notifications

- How to get a notification if the variable has changed?



- In the component declaration:

```
derive from oc::PropertyListener
```

è Listener Method 'PropertyHasChanged' (gets called by GUI)

- In the component's code:

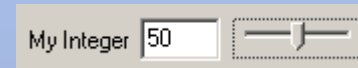
```
GetProperties()["Parameter:MyInteger"].AddListener(this);
```

- In PropertyHasChanged:

```
void PropertyHasChanged(oc::Property& prop,
 bool about_to_be_deleted)
{
 if(about_to_be_deleted) return;
 if(prop.GetName()=="Parameter:MyInteger") {
 // do something
 }
}
```

# Example: feedback to GUI

- `AddProperty( "Parameter:MyInteger" , mMyInt )`  
...  
`mMyInt = 51;`



- Problem: changing the variable by accessing `mMyInt` directly is not noticed by GUI
- How to notify the GUI about changes of a variable?

```
GetProperties()["Parameter:MyInteger"]=51;
```

*This will result in a exception if the type of the parameter and the type of the variable assigned do not match!*

# GUI file <rules>

---

- Structure:

```
<rule>
 <if>
 <condition> s.th. that evaluates to true or false</condition>
 <action> list of assignments </action>
 </if>
 ... elseif, else
</rule>
```

- <condition>:

Parameter:Name.state operator value (list/combination of values)

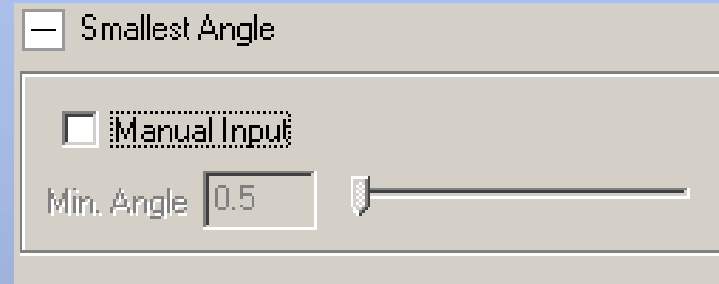
- operator can be ==, LT,GT,LE,GE
- value can be single value, range, regular expression
- list of values are comma separated
- values can be combined using bracketing, AND, OR

- <action>:

Parameter:Name.state = value

# GUI file <rules>: enable

- Enable/Disable example:



```
<rule>
 <if>
 <condition>Parameter:MI.value==true</condition>
 <action>Parameter:MinA.enable = true</action>
 </if>
 <else>
 <action>Paramter:MinA.enable = false</action>
 </else>
</rule>
```

- Examples à see TestPanel in SimTest application

# GUI file <rules>: defer/trigger

- Defer/Trigger mechanism

If the defer flag of a parameter is set, the variable is not automatically updated. The variable is only updated when the trigger flag is set.

- Defer/Trigger example:

```
<action>Parameter:MinA.defer = true</action>
<rule>
 <if>
 <condition>Parameter:p1.value== 2, 4, "[3-9]",
 3-4 OR (Parameter:p2.value LT Parameter:p3.value)
 </condition>
 <action>Parameter:MinA.trigger = true</action>
 </if>
</rule>
```

# OrcanWx

---

- based on wxWidgets ( à [www.wxwidgets.org](http://www.wxwidgets.org))
- 3 main OrcanWx classes:
  1. DynamicParameterPanel
    - Automatically manages all variables of a component's PropertyMap (ORCAN's reflexive container)
  2. DynamicParameterDialog
    - Like DynamicParameterPanel in a dialog
  3. DynamicLayout
    - Easy way to layout a group of child windows

# OrcanWx – Minimal Application

```
#include <wx/wxprec.h>
#ifdef WX_PRECOMP
include <wx/wx.h>
#endif
```

← wxWidgets includes

```
#include <oc/ObjectBroker.hh>
#include <ocs/Radiation.hh>
#include <ocwx/DynamicParameterPanel.hh>
```

← ORCAN includes

```
class MyApp: public wxApp
{
 bool OnInit()
 {
 ocs::RadiationRef rad = ocs::Radiation::New();
 if(!rad)
 { OCERROR("could not create Rad instance"); return false; }

 wxFrame* frame = new wxFrame(NULL,-1,wxT("MyGUI"));
 wxPanel* params = new ocwx::DynamicParameterPanel(
 frame, -1, wxPoint(0, 0), wxSize(350, 400), wxDEFAULT, "rad",
 rad.GetProperties());
 frame->Show(TRUE);

 return true;
 }
};
```

← wxWidgets main

```
IMPLEMENT_APP(MyApp)
```

← gen. wxWidgets code



# OrcanWx – Minimal Application

```
#include <wx/wxprec.h>
#ifdef WX_PRECOMP
#include <wx/wx.h>
#endif
```

wxWidgets includes

```
#include <oc/ObjectBroker.hh>
#include <ocs/Radiation.hh>
#include <ocwx/DynamicParameterP
```

ORCAN includes

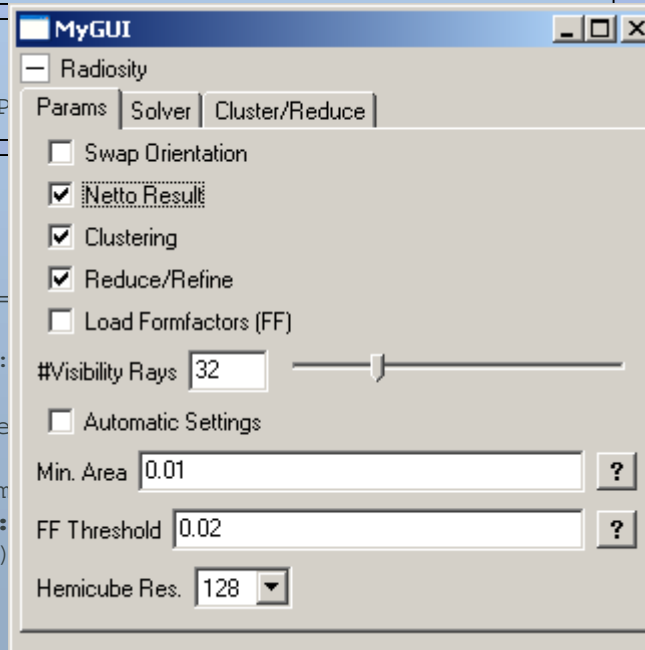
```
class MyApp: public wxApp
{
 bool OnInit()
 {
 oc::ObjectBroker & obroker =

 ocs::RadiationRef rad = ocs:
 if(!rad)
 { OCERROR("could not create

 wxFrame* frame = new wxFram
 wxPanel* params = new OCWX::
 frame, -1, wxPoint(0, 0)
 rad.GetProperties());
 frame->Show(TRUE);

 return true;
 }
};
```


wxWidgets main



```
IMPLEMENT_APP(MyApp)
```

gen. wxWidgets code

# Runtime GUI Modification

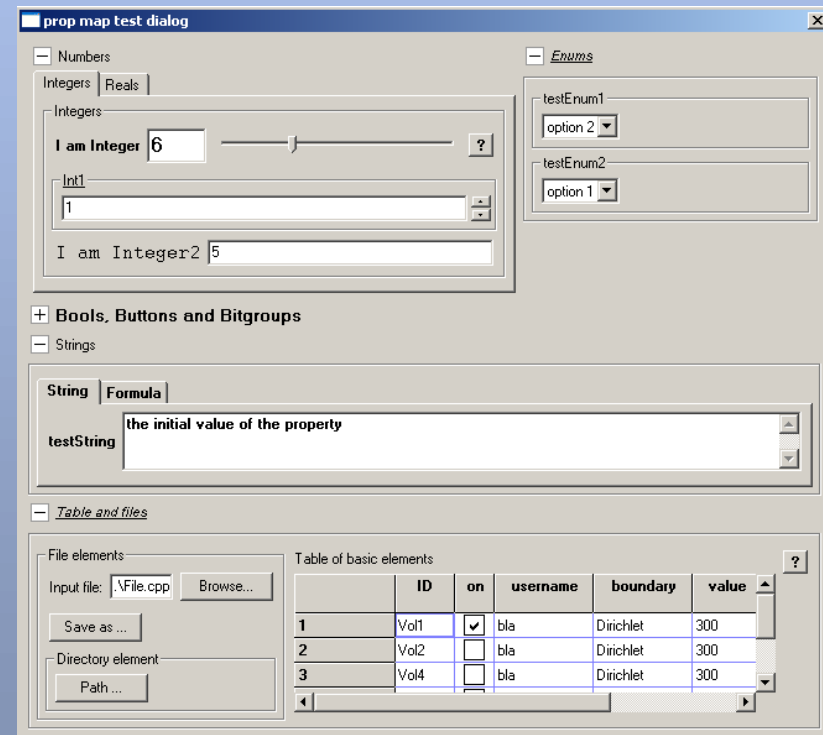
- All aspects of a GUI element (its state) can be manipulated by the components (value, enable, visible, readonly, domain, layout, notify, trigger, defer)
- The state of a GUI element is stored in an extra 'resource' attached to each parameter
- Example: set domain of a real A screenshot of a GUI slider control. It consists of a small text box on the left containing the number "300", followed by a horizontal track with a vertical slider knob on the right side.

```
std::stringstream contour_domain;
contour_domain << "<real><start>" << mMapScalarMin
 << "</start><end>" << mMapScalarMax
 << "</end><step>" <<
 (mMapScalarMax-mMapScalarMin)/100.
 << "</step></real>" << std::ends;
```

```
GetProperties()["Parameter:ContourValue"].GetResource().SetState(
 "domain", contour_domain.str());
```

# GUI Reference

- doc/README\_GUI.txt
- SimTest's TestPanel →

**prop map test dialog**

**Numbers**

Integers | Reals

Integers

I am Integer 6

Int1  
1

I am Integer2 5

**Enums**

testEnum1  
option 2

testEnum2  
option 1

**Bools, Buttons and Bitgroups**

**Strings**

String | Formula

testString the initial value of the property

**Table and files**

File elements

Input file: .\File.cpp Browse...

Save as ...

Directory element

Path ...

Table of basic elements

	ID	on	username	boundary	value
1	Vol1	<input checked="" type="checkbox"/>	bla	Dirichlet	300
2	Vol2	<input type="checkbox"/>	bla	Dirichlet	300
3	Vol4	<input type="checkbox"/>	bla	Dirichlet	300