SCIRun: Module Development Basics

CIBC/NEU Workshop 2012

http://bit.ly/SCIRunDevWorkshop

Goals

- Take you from "Hello World" in SCIRun to being able to develop an interesting module.
- Learn some software engineering best practices along the way.
- Build some excitement for the next version coming in 2013.

Prerequisites

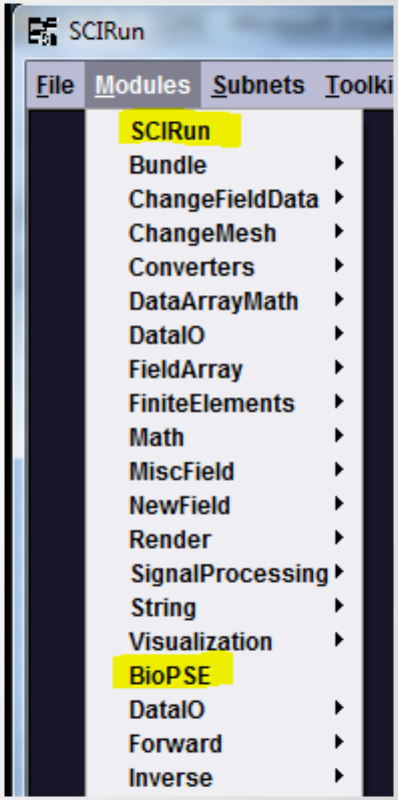
- Understand basics of object-oriented development in C++ (class, method, inheritance)
- C++ development tools: compiler, linker
- Downloaded SCIRun source code
  - http://bit.ly/SCIRunSource4_6
- CMake (can create XCode projects, GNU make, NMake, Visual Studio)
- GNU make on Linux/Unix platforms

Build Instructions

http://www.sci.utah.edu/devbuilds/scirun_docs/DeveloperGuide.pdf

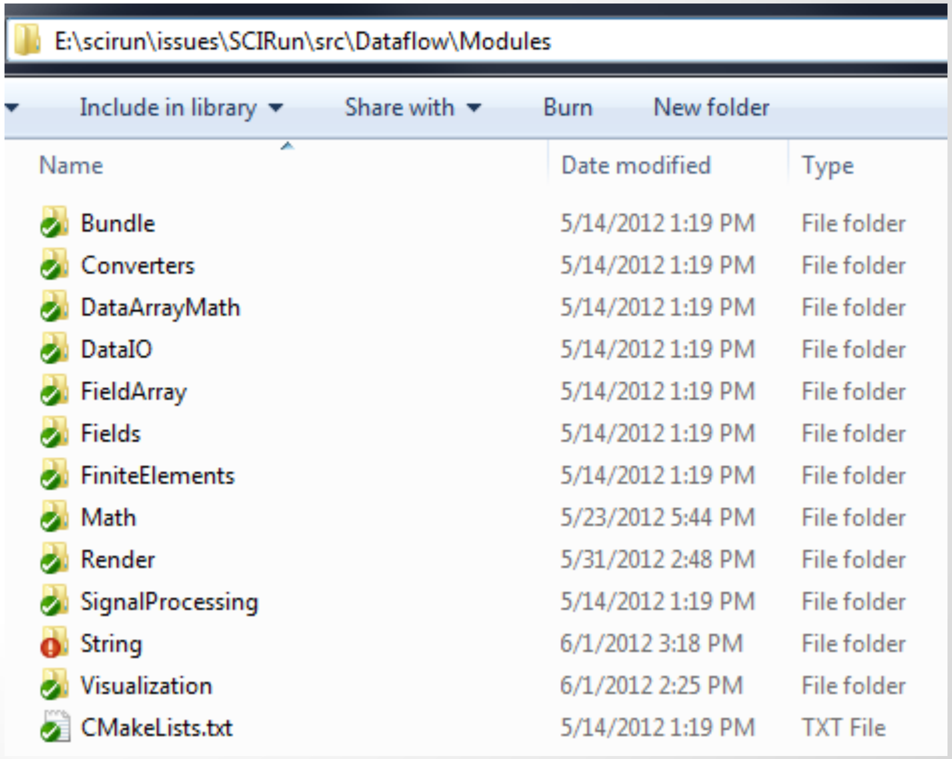
Module specification: Packages

- Packages are the SCIRun plugin mechanism, and each Package gets its own menu item on the Network Editor. Core modules in SCIRun live in the SCIRun Package.
  - In your working tree, the SCIRun Package corresponds to everything under the SCIRun/src/Dataflow/Modules directory.
  - All other Packages live in SCIRun/src/Packages.



Module specification: Categories

Categories are the directories under Modules. These are also reflected in the menu item, and should contain modules that share some common ground defined by the category.

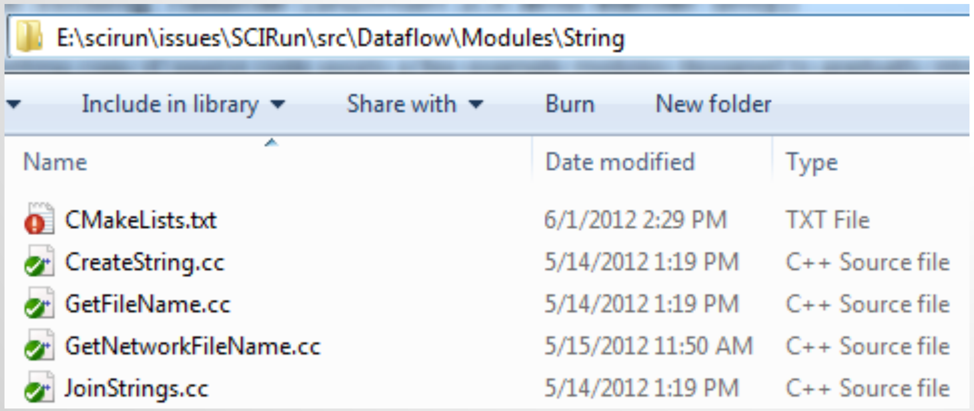


E:\scirun\issues\SCIRun\src\Dataflow\Modules

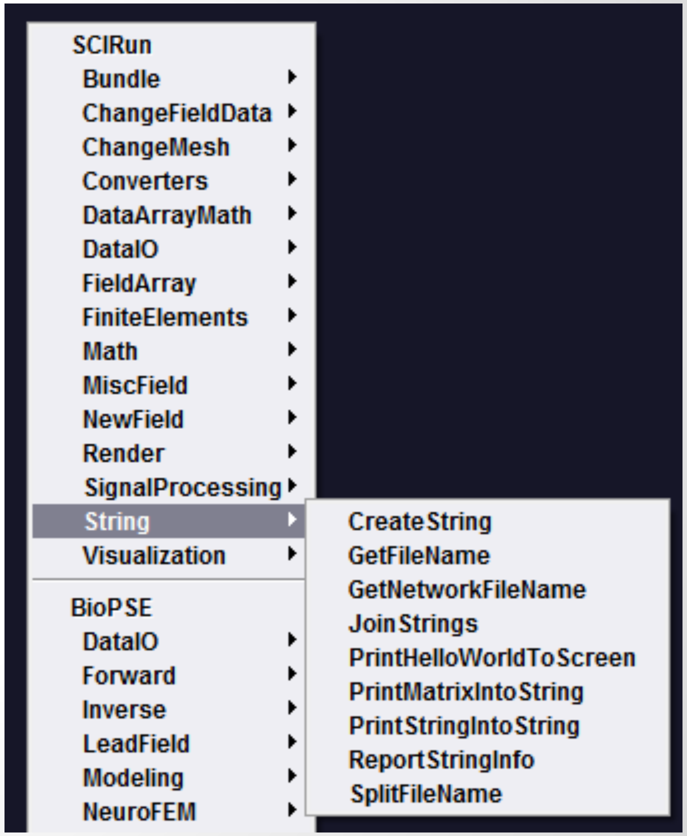
Name	Date modified	Type
Bundle	5/14/2012 1:19 PM	File folder
Converters	5/14/2012 1:19 PM	File folder
DataArrayMath	5/14/2012 1:19 PM	File folder
DataIO	5/14/2012 1:19 PM	File folder
FieldArray	5/14/2012 1:19 PM	File folder
Fields	5/14/2012 1:19 PM	File folder
FiniteElements	5/14/2012 1:19 PM	File folder
Math	5/23/2012 5:44 PM	File folder
Render	5/31/2012 2:48 PM	File folder
SignalProcessing	5/14/2012 1:19 PM	File folder
String	6/1/2012 3:18 PM	File folder
Visualization	6/1/2012 2:25 PM	File folder
CMakeLists.txt	5/14/2012 1:19 PM	TXT File

Module Specification: Modules

Finally under each Category directory are the Modules contained within them. Each module has a . cc file with the same name as the module.


E:\scirun\issues\SCIRun\src\Dataflow\Modules\String

Name	Date modified	Type
CMakeLists.txt	6/1/2012 2:29 PM	TXT File
CreateString.cc	5/14/2012 1:19 PM	C++ Source file
GetFileName.cc	5/14/2012 1:19 PM	C++ Source file
GetNetworkFileName.cc	5/15/2012 11:50 AM	C++ Source file
JoinStrings.cc	5/14/2012 1:19 PM	C++ Source file



Creating your first module

"Hello World!" -- SCIRun style.

You'll need three files:

- SCIRun/src/Dataflow/Modules/String/PrintHelloWorldToScreen.cc
- SCIRun/src/Dataflow/XML/PrintHelloWorldToScreen.xml
- SCIRun/src/Dataflow/Modules/String/CMakeLists.txt

"Hello World" Example: C++ Source

SCIRun/src/Dataflow/Modules/String/PrintHelloWorldToScreen.cc

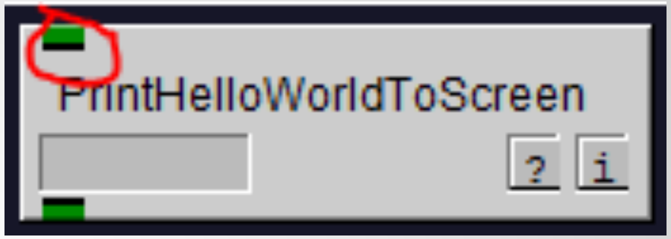
"Hello World" Example: XML Description

SCIRun/src/Dataflow/XML/PrintHelloWorldToScreen.xml

"Hello World" Example: Adding files to project

SCIRun/src/Dataflow/Modules/String/CMakeLists.txt

Getting Input: Overview



Note: if you are writing a module as we go, you could simply add the new bits as you go to your module.

Now that we know how to execute, we would like to get some data passed into this module through its input port. First we need to let SCIRun know that we expect input and what type of input we expect. This happens in the XML file.

Getting Input: XML description

```
<io>
   <inputs lastportdynamic="no">
     <port>
      <name>StringToPrint</name>
      <datatype>SCIRun::String</datatype>
     </port>
   </inputs>
 </io>
```

The io section of our component contains port information. We give it a name (important to remember as we ask for this port by name from our C++ code) and we also give it a datatype. In this case we want to and only will accept SCIRun::String as an input. Now when we instantiate this module, it will be created with a String input port.

Getting Input: C++ Implementation

```
void PrintHelloWorldToScreen::execute()
{
  StringHandle input;
  get_input_handle("StringToPrint", input, false);
  const std::string greeting = "Hello World! " +
       (input.get_rep() ? input->get() : "");
  std::cerr << greeting << std::endl;
}
```

All data that pass through ports in SCIRun are passed using Handles. Our handles are reference counted locking handles. They are essentially smart pointers that delete the object they point to when no one has a reference to it anymore.

Getting Input: Dataflow Details

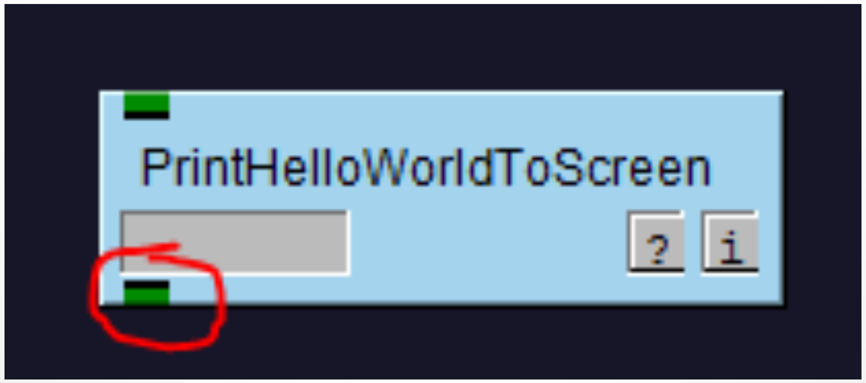
```
StringHandle input;
get_input_handle("StringToPrint", input, false);
```

First we declare an empty FieldHandle, then we ask for it to be filled with data from the input port named "InField". As you recall this is the exact string used in the xml file to declare the port name.

get_input_handle does a number of things for you. If the last argument is true, then we require input data on this port (some ports are optional). get_input_handle then blocks waiting for data to arrive on the port, makes sure it is non NULL, and sets the handle appropriately.

Sending Output: Overview

Essentially the dual of getting input, as you'd expect.



Sending Output: XML description

```
<io>
  <inputs lastportdynamic="no">
    <port>
     <name>StringToPrint</name>
     <datatype>SCIRun::String</datatype>
    </port>
  </inputs>
  <outputs>
    <port>
     <name>Greeting</name>
     <datatype>SCIRun::String</datatype>
    </port>
  </outputs>
 </io>
```

Sending Output: C++ implementation

```
void PrintHelloWorldToScreen::execute()
{
  StringHandle input;
  get_input_handle("StringToPrint", input, false);
  const std::string greeting = "Hello World! " + (input.
get_rep() ? input->get() : "");
  std::cerr << greeting << std::endl;
  StringHandle output(new String(greeting));
  send_output_handle("Greeting", output);
}
```

send_output_handle takes the exact string with which we declared the output port in our xml file, and we send the handle to our field along through the output port.

Module I/O: Improvements for next version

- Less reliance on hard-coded strings in multiple places--easy for them to get out of sync, cause subtle bugs.
- Encode I/O signature in module class itself
  - Think C/C++ function prototype and its accompanying type safety and discovery
    The signature gives you an exact description, and a module has the same exact property: fixed input/output number and types
  - C++ language limits the implementation options
- More functional/natural style of sending/receiving data values
  - Think of piping values on the unix command line

Changing Data Values: Algorithm Classes

- SCIRun contains Algorithm libraries corresponding to its many Module libraries
  - The idea is to separate the GUI-dependent code from the algorithm behind each module's execution
- A basic layering of SCIRun program logic becomes apparent:
  - GUI interaction: receive input from user
  - Module execution: manage ports, connections, input validation and passing choices to algorithms
  - Algorithmic code: math, geometry
  - Datatypes: basic building blocks

Low-level libraries: threading, persistence, etc

Test-driving your algorithm independently from the GUI: Good Time for a Unit Test

- Main idea: write test code that can be run automatically to verify your algorithm, without needing to call up a GUI or read a local file
  - Most essential for brand new code--more difficult for older code or code written by others
  - Also serves as up-to-date documentation: wiki pages can grow stale, but test code, included in the project, will always need to compile and run
- For those interested in this software engineering idea (write bug-free code!), examples are available using Google's Testing framework (start in directory SCIRun\src\Core\Datatypes\Tests)

User Input: Writing Basic Dialogs

NOTE: this process will completely change in SCIRun v5.0. If you're interested in these ongoing developments, let us know and we can have a separate meeting about.

Thus, we will not show the current Tcl/Tk methodology for writing GUIs at this session. Please consult Ayla or your local SCIRun expert.

Conclusion

- You have the basic structure of a module now:
  - Port description via XML file
  - Override Module::execute
  - get_input_handle/send_output_handle
- What goes in between is up to your imagination
  - SCIRun contains over 400 modules at the moment, so there's plenty to work with and learn from
  - Even with 15+ years of existence, this code is open to fresh ideas

Thank you

dwhite@sci.utah.edu