GreenConfig

Declare parameters in user modules

#include <greencontrol/config.h>

SC\_MODULE(module\_name) {

gs\_param<type> param\_name;

SC\_CTOR(module\_name): param\_name("param\_name") {

}

}

Read a text configuration file named “example.cfg”

#include <greencontrol/config\_api\_config\_file\_parser.h>

int sc\_main(int argc, char \*argv[]) {

gs::cnf::ConfigFile\_Tool configreader("configreader");

configreader.config("example.cfg");

// look for cmd line option --gs\_configfile *<file>*

configreader.parseCommandLine(argc, argv);

}

Read a Lua configuration file named “example.lua”

#include <greencontrol/config\_api\_lua\_file\_parser.h >

int sc\_main(int argc, char \*argv[]) {

gs::cnf::LuaFile\_Tool luareader("luareader");

luareader.config("example.lua");

// look for cmd line option --gs\_luafile *<file>*

luareader.parseCommandLine(argc, argv);

}

Parse command line to set individual parameters

#include <greencontrol/config\_api\_command\_line\_parser.h>

int sc\_main(int argc, char \*argv[]) {

gs::cnf::CommandLineConfigParser cmdlineparser("cmdlineparser");

// look for cmd line option --gs\_param *<name>*=*<value>*

cmdlineparser.parse(argc, argv);

}

Refer to a parameter in another module

#include <greencontrol/config.h>

my\_function\_or\_method() {

gs::cnf::cnf\_api \*m\_configAPI = gs::cnf::GCnf\_Api::getApiInstance(NULL);

gs::gs\_param\_base other\_param = m\_configAPI->getPar("Other.param");

std::cout << other\_param->getString() << std::endl;

}

Register callback function

#include <greencontrol/config.h>

SC\_MODULE(module\_name) {

GC\_HAS\_CALLBACKS();

SC\_CTOR(module\_name) {

gs::cnf::cnf\_api \*m\_configAPI = gs::cnf::GCnf\_Api::getApiInstance(this);

m\_configApi->REGISTER\_NEW\_PARAM\_CALLBACK(module\_name, new\_param\_cb);

GC\_REGISTER\_PARAM\_CALLBACK(someparam,module\_name,someparam\_cb);

}

~module\_name () {

GC\_UNREGISTER\_CALLBACKS();

}

new\_param\_cb(const std::string name, const std::string val) {

cout << "New parameter " << name << endl;

gs::gs\_param\_base \*par = m\_configApi->getPar(parname);

cout << " of type " << par->getTypeString() << endl;

}

someparam\_cb(gs::gs\_param\_base& par) {

if (!par.is\_destructing()) {

cout << par.getName() << " changed to " << par.getString() << endl;

}

}

}

Unregister a callback function

gs::cnf::ParamCallbAdapt\_b\* someparam\_cb\_handler =

GC\_REGISTER\_PARAM\_CALLBACK(someparam, module\_name, someparam\_cb);

GC\_UNREGISTER\_CALLBACK(someparam\_cb\_handler);

Using environment variables to set parameters

1) can be used anywhere a param is set from a string

2) syntax $(varname)

3) the escape sequence is to double the dollar ($) sign

Declare a parameter array

#include <greencontrol/config.h>

SC\_MODULE(module\_name) {

gs\_param<int\*> arrInt;

my\_method {

arrInt.setString("{0 1 2 3 4 5 6 7 8 9}");

cout<<"arrInt size="<< arrInt.size() <<" values=" << arrInt.getString();

arrInt.resize(15);

arrInt[12] = 12;

arrInt.at(20) = 20; // automatic resize

}

}

Declare extended parameter arrays (static or dynamic)

#include <greencontrol/config.h>

SC\_MODULE(module\_name) {

gs::gs\_param\_array arrayTop;

gs::gs\_param\_array array1;

gs::gs\_param<int> array1\_intPar;

gs::gs\_param\_array \*array2;

gs::gs\_param<string> \*array2\_strPar;

SC\_CTOR(module\_name)

: arrayTop("arrayTop")

, array1("array1", arrayTop)

, array1\_intPar("array1\_intPar", 123, array1)

{

array2 = new gs::gs\_param\_array("array2", arrayTop);

array2\_strPar= new gs::gs\_param<string>("array2\_strPar","hello",array2);

}

}

Using private parameters

#include <greencontrol/config.h>

SC\_MODULE(module\_name) {

gs::param<int> pubParam1;

SC\_CTOR(module\_name)

: m\_privApi(this, "pubParam1", "child.other", END\_OF\_PUBLIC\_PARAM\_LIST)

, pubParam1("pubParam1")

, privParam1("privParam1")

{}

protected:

gs::cnf::GCnf\_private\_Api m\_privApi;

gs::param<int> privParam1;

}

GreenAV

List of output plugins provided by GreenSocs

|  |  |
| --- | --- |
| **Identification** | **Header file to #include** |
| gs::av::DEFAULT\_OUT | greencontrol/analysis.h |
| gs::av::NULL\_OUT | greencontrol/analysis.h |
| gs::av::TXT\_FILE\_OUT | greencontrol/analysis\_file\_outputplugin.h |
| gs::av::STDOUT\_OUT | greencontrol/gav/plugin/Stdout\_OutputPlugin.h |
| gs::av::CSV\_FILE\_OUT | greencontrol/analysis\_csv\_outputplugin.h |
| gs::av::SCV\_STREAM\_OUT | greencontrol/analysis\_scv\_outputplugin.h |
| gs::av::VCD\_FILE\_OUT | greencontrol/analysis\_vcd\_outputplugin.h |
| gs::av::TXT\_TD\_FILE\_OUT | greencontrol/gav/plugin/FileWithTd\_OutputPlugin.h |
| gs::av::VCD\_TD\_FILE\_OUT | greencontrol/gav/plugin/VCDWithTd\_OutputPlugin.h |

Declare GreenAV API inside a module and add a parameter to an output plugin

#include <greencontrol/analysis\_file\_outputplugin.h>

SC\_MODULE(module\_name) {

gs::av::GAV\_Api m\_analysisAPI;

SC\_CTOR(module\_name) {

m\_analysisAPI.add\_to\_default\_output(gs::av::TXT\_FILE\_OUT, someparam);

}

}

Create an output plugin instance (other than default) and add parameter to it

gs::av::OutputPlugin\_if\* csvFileOP =

m\_analysisAPI.create\_OutputPlugin(gs::av::CSV\_FILE\_OUT, "CSVexample.log");

csvFileOP->observe(someparam);

m\_analysisAPI.add\_to\_output(csvFileOP, other\_param);

Instantiate trigger on parameter, event or interval

gs::gs\_param<bool> triggerParam("triggerParam");

gs::av::Trigger trigger1(triggerParam);

sc\_event triggerEvent;

gs::av::Trigger trigger2(triggerEvent);

gs::av::Trigger trigger3(10, SC\_NS);

// methods: enable\_on\_change\_activation() and disable\_on\_change\_activation()

Instantiate a calculator and set formula

gs::av::Calculator<int> c1("c1");

c1.calc("/", c1.calc("+" int\_par, uint\_par), 2);

c1.enable\_sliding\_window(5); // average the last 5 results

Instantiate statistics calculator using a trigger and a calculator, add to output

gs::av::StatCalc<int> statCalc1("statCalc1", trigger, calc);

gs::av::StatCalc<int> statCalc2("statCalc2", calc); // default trigger

m\_analysisAPI.add\_to\_default\_output(gs::av::STDOUT\_OUT,

statCalc.get\_result\_param());

// control activation with activate() and deactivate() methods

// calculate\_now() method works only when active

Report Messages

List of message configuration members (defaults to NULL/false)

|  |  |  |
| --- | --- | --- |
| **C++ Type** | **Member name** | **Notes** |
| string | msgconfig\_name | filename or special output name |
| bool | msgconfig\_starttime\_en | enable output in a time interval |
| sc\_time | msgconfig\_starttime\_en | Time to start the output |
| sc\_time | msgconfig\_endtime\_en | Time to end the output |
| debug\_msg\_level | msgconfig\_dbglvl | maximum debug level (verbosity) to output |
| bool | msgconfig\_info\_en | Output sc\_info messages? |
| bool | msgconfig\_warn\_en | Output sc\_warning messages? |
| bool | msgconfig\_error\_en | Output sc\_error messages? |
| bool | msgconfig\_fatal\_en | Output sc\_fatal messages? |
| bool | msgconfig\_printtime | Prepend simulation time? |
| bool | msgconfig\_printname | Prepend stream name? |
| bool | msgconfig\_printfile | Prepend C++ file source? |
| bool | msgconfig\_printlevel | Prepend verbosity level? |
| vector<string> | msgconfig\_module\_id | list of modules/streams to be captured (all if empty) |

Creating and applying a message configuration in the code

gs::report::msg\_configuration cnf;

cnf.msgconfig\_name = "report\_file.txt";

cnf.msgconfig\_dbglvl = gs::report::dbg\_msg\_L9;

cnf.msgconfig\_printfile = true;

cnf.msgconfig\_module\_id.push\_back("ModuleA"); // ModuleA

cnf.msgconfig\_module\_id.push\_back(""); // top-level

gs::report::MessageStreamer::apply\_configuration(cnf);

A classic config file defining a message configuration

MessageStreamer\_config.0.msgconfig\_name "report\_debug.txt"

MessageStreamer\_config.0.msgconfig\_dbglvl 9

MessageStreamer\_config.0.msgconfig\_printfile true

MessageStreamer\_config.0.msgconfig\_module\_id "{"ModuleA", "ModuleB"}"

A lua config file defining message configuration

MessageStreamer\_config = {

{

msgconfig\_name = "report\_system.txt",

msgconfig\_dbglvl = 0,

msgconfig\_info\_en = true,

msgconfig\_warn\_en = true,

msgconfig\_error\_en = true,

msgconfig\_fatal\_en = true,

msgconfig\_printtime = true,

msgconfig\_printname = true,

msgconfig\_printfile = false,

msgconfig\_printlevel = false

msgconfig\_module\_id = "{\"ModuleA\", \"ModuleB\"}"

}

}

Defining message streams in a module

#include "greencontrol/reportmsg/gs\_debug\_stream.h"

#include "greencontrol/reportmsg/gs\_system\_stream.h"

SC\_MODULE(module\_name) {

gs::report::gs\_debug\_stream dbgL2;

gs::report::gs\_system\_stream sysINFO;

SC\_CTOR(module\_name)

: dbgL2("dbgL2", gs::report::dbg\_msg\_L2)

, sysINFO("sysINFO", gs::report::sys\_msg\_INFO)

{}

}

Using the report messages

dbgL2 << "This is debug level 2" << std::endl

<< "other line for the same message" << GS\_END\_MSG;

sysINFO << "This is an sc\_info in just one line" << GS\_END\_MSG;