GreenControl 4.x Quickreference Card

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;
```