Getting Started with SansGUI®

A Quick SansGUI Tutorial for Simulation Users and Developers

All rights reserved.
What is SansGUI ...

- An Interactive Environment
  - for building models, entering parameters, running simulations, monitoring progress, and viewing results

- A Software Framework
  - for developing and deploying simulation programs without tedious Graphical User Interface programming
Who are the users …

Anyone who is involved in developing and using computer simulation in scientific and engineering fields.

- **Simulation Users**
  - Use SansGUI Run-Time Environment

- **Simulation Developers**
  - Use SansGUI Development Environment
Simulation Users

Those who use building blocks to configure experimental models, perform simulation runs, study the effects from different sets of inputs in order to fine tune their designs.

- Research Assistants
- Project Engineers
- Design Engineers
- Data Analysts
Simulation Developers

Those who study the underlying logic and mathematics of target systems to define model building blocks and implement computer algorithms for simulation.

- Research Scientists
- Research Engineers
- Software Developers
SansGUI Architecture

- **SansGUI Development Environment**
  - Define model building blocks in classes
  - Specify attributes involved in simulation model
  - Implement simulator logic and mathematics

- **SansGUI Run-Time Environment**
  - Configure model and prepare input data
  - Run simulation and monitor progress
  - Analyze simulation results to refine design
Run–Time Features

- Interactive Model Building Tools
- Data Entry Assistance and Validation
- User Extensible Unit Conversion
- Model Data Version Synchronization
- Interactive Simulation Control
- Dynamic Charting & 3D Animation
- User Plug-In Routine Support
Development Features

- Interactive Simulator Development
- Class Schema Definition & Update
- Programmable Data Validation
- Unit Object Creation & Maintenance
- SansGUI Source Code Framework
- Interactive Tracing and Debugging
- OpenGL® 3D Graphics Programming
Using SansGUI®
Modeling and Simulation

- Working with SansGUI
  - System Abstraction - Entity / Relation
  - Model Configuration
  - Data Entry / GUI Control Types
  - User Extensible Unit Conversion
  - Simulation Run Controls
  - Simulation Result Logging and Plotting
  - Animated 3D Graphics Controls
A Model of the Target System

denotes physical links with directionality; non-directional links are without arrows

denotes referential links
Components and References

Component Objects
- Physical objects used to create parts
- Links are special components that connect parts to form network models

Reference Objects
- Informational objects referred to by other objects, parts or links
- Collection, Graphics, Matrix, Table, and more
Class Hierarchy

Object-Attribute-Value

Class-A
- Diameter (mm)
- Length (mm)

Class-B
- Diameter (mm)
- Length (mm)
- Curvature (mm)
- Orientation

Object-1
- Diameter = 1.2 in
- Length = 10.5 in
- Curvature = 5.0 in
- Orientation = "SE"

Object-2
- Diameter = 2.0 in
- Length = 4.8 in
- Curvature = 10.0 in
- Orientation = *

Object-3
- Diameter = 2.5 in
- Length = 9.0 in
- InnerDiameter = 2.0 in

* see next slide
System Parts List

Overriding Values

Object-1
- Diameter = 1.2 in
- Length = 10.5 in
- Curvature = 5.0 in
- Orientation = "SE"

Part-1

Object-2
- Diameter = 2.0 in
- Length = 4.8 in
- Curvature = 10.0 in
- Orientation = *

Part-2

Part-3
- Length = 14.0 in

Part-4
- Orientation = "NE"
Assembly Hierarchy

TOP Assembly

Part

Part

Subassembly

Part
Project Models

- **Class Hierarchy**
- **Assembly Hierarchy**
Project Models

- Class Hierarchy
- Assembly Hierarchy
Model Configuration

- Common Properties in Objects
  - Start from Simulator Object Libraries

- Hierarchical Assemblies of Parts
  - Create parts and links from objects
  - Form assemblies with parts and links
  - Export ports to parent assembly for linking
  - Replicate parts and all its subassemblies
  - Override object values in parts or links
Data Entry/GUI Control Types

- **Numbers**
  - Integer
  - Floating Point (single or double precision)
  - Currency Format

- **String**
  - Regular
  - Masked
  - Encrypted

- **URL / File**
- **Date / Time**
- **On-Off Switch**
- **Tri-State Switch**
- **Enumerated Items**
- **Object Reference**
- **Quality**
- **Symbolic Parameter**
Unit Conversion

- **User Extensible Unit Objects**
  - Specify units of measure for data input, output and presentation
  - Automatic conversion to units required by the simulators before simulation runs
  - Expand unit tables to add new units not covered by the simulator developer
  - Service Session - lock / unlock unit objects by simulation developers
Simulation Run Controls

**In-Process Simulation Controls**
- Run, pause / resume, step, fast forward, stop
- Monitor and change data values on the fly
- Simulation runs in a thread within SansGUI

**External Process Controls**
- Run and stop a simulation
- Customize invocation script
- Simulation runs in a separate process
Simulation Control Objects

- **In-Process Cycle Simulation Control**
  - Continuous, cycle-driven simulation control
    - Program location, log file name, start / pause / end cycle number, current cycle number, screen refresh interval, part evaluation order

- **External Process Simulation Control**
  - Stand-alone or legacy code integration
    - Program location, invocation script, model file name and type, working directory, command parameters
Results Logging and Plotting

- **Logging**
  - Select logged values in objects, parts or links
  - Logged results can be viewed in a data grid
  - Change output units as desired

- **Plotting**
  - Plot selected set of logged data when running
  - Change plot specifications when paused
  - Customize plotting features
SansGUI Object Library

- Simulator Dependent
- Class Schema Version Control
- Convenient & Custom Objects
  - Mandatory Objects: required by simulators
  - Default Objects: with default values
  - Common Objects: frequently used values
  - Special Objects: with hard to enter or remember values
Run-Time Environment

SansGUI Run-Time Environment

- SansGUI Object System
- Model & Schematic Editor
- Version Control
- Execution Control
- Result Logger & Viewer

Simulation User

Object Library
- .SGO File

On-Line Help
- .HTM, .HLP, etc.

Project Model
- .SGP File

Model File
- .TXT or .XML

On-Line Help
- .HTM, .HLP, etc.
Run-Time Environment

Object Library
- .SGO File

On-Line Help
- .HTM, .HLP, etc.

User Override Routines

SansGUI Run-Time Environment
- SansGUI Object System
- Model & Schematic Editor
- Version Control
- Execution Control
- Result Logger & Viewer

In-Process
- .DLL File
  - Editing
  - Execution
  - Evaluation

Invocation Script
- .BAT File

Project Model
- .SGP File

Model File
- .TXT or .XML

External Process
- Local or Remote

Software Implementation
- Core Simulator

Simulation User
Advanced Simulation Users

- Managing Common Objects
  - Customizing Object Libraries
  - Importing Objects and Classes

- Overriding Simulator Routines

- Customizing SansGUI Environment
  - Managing User Workspaces
  - Tuning Environment Parameters
  - Using Different Directories
Developing with SansGUI®
Simulator Development

- SansGUI File Types and Architecture
- Simulator Schema Definition
- Input Data Specification / Validation
- Internal Data Structures
- Core Simulator Programming
- Pre-processors / Post-processors
- On-Line Help Development
SansGUI File Types

**Primary Extensions**

- **Model File**
  - `.TXT` or `.XML`

- **Schema Definition**
  - `.SGS`

- **Object Library**
  - `.SGO`

- **Project Model**
  - `.SGP`

- **Class Functions**
  - `.DLL`

- **User Functions**
  - `.DLL`

- **Legacy Simulator**
  - `.EXE`

- **External Process Simulator**

- **In-Process Simulator**

-> *denotes the source information is used to generate the target file or code framework*

-> *denotes the source information is sent to the target for execution*
SansGUI Development Environment

- Meta-Schema
- Object System
- Version Control
- Source Framework
- Schema Compiler

Schema Definition
.SGS File

Object Library
.SGO File

Simulation Developer

Development Environment
Development Environment

SansGUI Development Environment

- Meta-Schema
- Object System
- Version Control
- Source Framework
- Schema Compiler

In-Process Source Files
C/C++ or Fortran

Programming Language - IDE

External Process Source Files

Other Programming Libraries

Text Editor

Simulation Developer

Schema Definition
.SGS File

Object Library
.SGO File

In-Process
.DLL File

External Process
.EXE File

Other Programming Libraries
Development Environment

SansGUI Development Environment

- Meta-Schema
- Object System
- Version Control
- Source Framework
- Schema Compiler

In-Process Source Files
C/C++ or Fortran

Programming Language - IDE

External Process Source Files

Other Programming Libraries

Text Editor

Help Authoring Tool

Files for Distribution

- Schema Definition
  .SGS File
- Object Library
  .SGO File
- In-Process
  .DLL File
- External Process
  .EXE File
- Invocation Script
  .BAT File (optional)
- On-Line Help
  .HTM, .HLP, etc.

Simulation Developer
SansGUI Schema Definition

- Simulator Identification
- Class Properties and Attributes
- Class Sharing Options
- DLL Function Specifications
- Port Specifications in Components
- Connectivity Specifications
- Unit Objects
SansGUI Intrinsic Classes

- Component Classes
  - Base, Assembly, and Link

- Reference Classes
  - Collection, Graphics, Matrix, and Table

- Simulation Control Classes
  - Cycle and XProc

- Unit Class
External Process Simulator

- Model File Format
  - Tabular Data Blocks
  - XML Model Data

- Invocation Script Customization
  - Input Filter - convert SansGUI file format
  - Pre & Post Processor - integrate execution
  - Job Control - submit long running job
  - Remote Execution - run simulation remotely
In–Process Simulator

■ Checking DLL Function Entry Points
  ● Define DLL entry points in Class Properties

■ Using Source Code Framework
  ● Generate skeleton code in C/C++ & Fortran

■ Working with MS Visual Studio
  ● Create DLL projects and insert source files
  ● Implement simulator logic and mathematics
  ● Compile, debug, and trace through code
SansGUI Object System

- SansGUI Data Object Format
  - SG_OBJ Data Structure
  - SG_VALU Data Structure
  - GUI Control Type Funneling

- SansGUI API
  - DLL Function Prototype
  - DLL Function Entry Points
  - Cycle Simulation Calling Sequence
SG_OBJ Data Structure

- SansGUI Schema Version
- Object Status and User Data
- Class Information
  - Class Name, Path, Version, and Attributes
- Object Information
  - Component Path and Serial Number
- SG_VALU Data Array
typedef struct SG_OBJ_tag {
    SG_CONST UINT nSGobjSchema;
    INT iStatus, iUserData;
    SG_CONST INT iNumVars;
    SG_VALU *SG_CONST zValues;
    SG_CONST INT iVerMajor, iVerMinor, iVerPatch, iVerBuild;
    SG_CONST UINT nCmpnNo;
    const TCHAR *SG_CONST cObjName, cCmpnName;
    SG_CONST TCHAR *SG_CONST cClassPath, cCmpnPath;
    const TCHAR *SG_CONST *SG_CONST sVarNames;
} SG_OBJ;
SG_VALU Data Structure

- **Data Type**
- **Dimension**
  - Size, columns, and rows
- **Data Array**
  - All values are in arrays
  - INT*, FLOAT*, DOUBLE*
  - TCHAR* (Dynamic TCHAR Array)
  - TCHAR** (String Array)
typedef struct SG_VALU_tag {
    SG_CONST UINT nType;
    SG_CONST INT iSize, iCols;
    INT iRows;
    union {
        void *SG_CONST vData;
        INT *SG_CONST iData;
        FLOAT *SG_CONST fData;
        DOUBLE *SG_CONST dData;
        TCHAR *SG_CONST cData;
        TCHAR *SG_CONST *SG_CONST sData;
    }
} SG_VALU;
GUI Control Type Funneling

- **SG VALU Data Storage**
  - Number → INT, FLOAT, DOUBLE Array
  - String → String Array
    - Regular string in a cell → Dynamic TCHAR Array
  - URL / File / Reference → String Array
  - Date / Time → INT Array
  - On-Off and Tri-State Switch → INT Array
  - Enumerated Items → INT Array
SG_SIM_FUNC Arguments

- adjObj
- adjObj
- adjObj
- simCtrl
- refObj
- refObj
- adjObj
- adjObj
- adjObj
- chgChild (Reserved)
- Other Arguments:
  - cMessage
  - cCommand
  - pOutFile
typedef SG_RET_CODE (SG_SIM_FUNC)(
    SG_OBJ *const self,
    SG_OBJ *const simCtrl,
    SG_OBJ *const chgChild,
    SG_OBJ *const refObjs[], const INT *const piRefObjs,
    SG_OBJ *const adjObjs[], const INT *const piAdjObjs,
    SG_OBJ *const lnkObjs[], const INT *const piLnkObjs,
    TCHAR *const cMessage, const INT iMsgLen,
    TCHAR *const cCommand, const INT iCmdLen,
    SG_FILE *const pOutFile
);

**DLL Function Return Value**

- **SG_R_OK** - success, continue simulation
- **SG_R_LMSG** - display a message to user
- **SG_R_PAUS** - pause and inquire user
- **SG_R_STOP** - error detected by simulator
- **SG_R_VERS** / **SG_R_SCHM** - version
- **SG_R_ERR** - error detected by SansGUI
- **SG_R_*** | 24 Bit Simulator Error Number
DLL Function Entry Points

- **Data Editing Functions**
  - End Edit Check
  - Resize/Load Data

- **Execution Functions**
  - Resize/Initialize Data
  - Begin/End Run and Case

- **Evaluation Functions**
  - Pre Evaluation/Evaluation/Post Evaluation
Cycle Simulation Sequence

(1) All Reference Objects and then all Parts
(2) All Parts and then all Reference Objects

- Init-Size (2) / Initialization (1)
- Begin Run / Case (1)
- Pre Evaluation (1)
- Evaluation and Post Evaluation (2)
- End Run / Case (2)
Evaluation Cycles

- **Reference Objects**
  - By Name

- **Parts in Assembly Tree - Depth First**
  - By Name
  - By Z-Order
  - By Horizontal Scan Lines
  - By Vertical Scan Lines
  - Random
Using Class Graphics

- Support OpenGL® 3D Graphics
  - Initialize - mapped to Begin Run function
  - Reshape - mapped to Pre Evaluation function
  - Display - mapped to Post Evaluation function
  - Select - mapped to Evaluation function

- Simplify 3D Graphics Operations
  - Rotate, Pan, Zoom, and Reset
  - Print, Copy to clipboard, Save to image files
Visual Fortran Support

- Select Column-Major Data Storage
- Use Generated Framework
  - Source files are generated as in C/C++
  - Choose Fortran or C/C++ implementation on a class-by-class basis
- Use Supplied Header File - SGdllf.h
- Dump SG_OBJC in Visual Fortran
  - Use the distributed SGdump.f code
Data Access in Fortran

```fortran
real*4, dimension(*) :: fVar
POINTER(PTR_fVar, fVar)
PTR_zValues = self%pzValues
PTR_fVar = zValues(index)%vData
```

- **Fortran Variable** `fVar`
- **POINTER** `PTR_fVar`
- **Array of SG_VALUs via PTR_zValues**
- **SG_OBJ** `pzValues`
Development Environment

- Microsoft Visual Studio®
  - Microsoft Visual C++ 6.0 (SP3+)
  - Compaq Visual Fortran 6.1+
  - Simulator code debugging and tracing
  - Multi-Threaded Execution Support

- OpenGL® 3D Graphics Programming

- XML Model Data Support

- Existing Code/Library Integration
Other Language Support

- **In-Process Simulation - PC**
  - Can create Win32 DLLs
  - Can be called from Microsoft Visual C++
    - Function names (length and case sensitivity)
    - Compatible function argument list
  - Can access C data structures with pointers

- **External Process Simulation**
  - Can create stand-alone programs
    - Batch command / Shell script invocation
Experiencing with SansGUI®
Hands-On Sessions

- **Visual Calculator for SansGUI**
  - Creating a simple graphical calculator with step-by-step instructions

- **MIDI Player for SansGUI**
  - Showing a legacy program and an in-process layer work in concert for dynamic charting

- **Mixer Example for SansGUI**
  - Building, loading and solving a system of linear equations
Thank you!

- **Visit Our Web Site**
  - http://protodesign-inc.com
  - http://sansgui.com

- **E-Mail**
  - Information: info@protodesign-inc.com
  - Sales: sales@protodesign-inc.com
  - Support: support@protodesign-inc.com
  - Beta testing: beta@protodesign-inc.com