SiSi 2.0

(**Si**mulation **S**ystem **I**nterface)

Documentation

Kai Reinhard[1] Martin Althaus[1]
Michael Müller[2] Uwe Bergmann[1]

July 31, 1998

SiSi grows up with its developers and users!

[1]Center for Environmental Systems Research Kassel
[2]Umweltforschungszentrum Leipzig

Contents

1 *SiSi* Files **2**
1.1 Input Files . . . 2
1.1.1 The syntax of a *SiSi* input file . . . 2
1.1.2 Example parameter files . . . 3
1.1.3 Reserved parameter names . . . 4
1.1.4 The syntax of a *SiSi* simulation file . . . 5
1.1.5 Example simulation file . . . 5
1.2 File Suffixes . . . 6

2 X*SiSi* **7**
2.1 Main Window . . . 7
2.2 Edit Options . . . 7
2.3 Edit Parameter . . . 8
2.4 Emit C++ Code . . . 9

3 Simulation development **11**
3.1 The easy way to include the *SiSi* fileparser . . . 11
3.2 Complete support by **X***SiSi* . . . 12
3.3 Different Compilers . . . 13
3.3.1 Gnu C++ Compiler . . . 13
3.3.2 Borland C++ Compiler . . . 14

4 FAQ **15**

5 Availability **16**
5.1 ftp server . . . 16
5.2 *SiSi*'s home page . . . 16

6 Copyright ©1997, 1998 **16**

List of Figures

1 Screenshot: Main Window . . . 7
2 Screenshot: Window for editing options . . . 7
3 Screenshot: Window for editing singular parameters . . . 8
4 Screenshot: Emit C++ Code Window . . . 9

Abstract

SiSi is an environment for simulations. A programmer could include many useful features of *SiSi*, such as a comfortable fileparser to read simulation parameters from files, data structures like strings, lists, tables and functions like round, gaussian distributed random numbers, ...
With **X***SiSi* there is a graphical user interface available for manipulating parameterfiles. So simulation programmers could make their simulations transparent to other users. The great advantage is, that this users doesn't have to understand the programming side of the simulation environment. They can use the simulation environment by manipulating the parameters with **X***SiSi*.
Now, it is very easy for the simulation programmer to include the *SiSi*-Library. One has only to add TWO lines to the source code (One for include *SiSi* header file and the other to call *SiSi* to read the parameter files). After, all variables are available as global. Writing result files will be realized very easy with few lines.

1 *SiSi* Files

SiSi (and also **X***SiSi*) owns a file parser for scanning input files. Input files will be generated by **X***SiSi*. The following sections describe the format of the input files supported by *SiSi*.

1.1 Input Files

1.1.1 The syntax of a *SiSi* input file

An input file consists of entries. An entry begins with a keyword specifiing the entry type, followed by the name of the entry, the header and the value.

Type keywords: The type keywords specify the type of the following entry.

`comment` A comment will be added to the parameter list. Commands only consist of the value. The value is the part beginning after `comment` until the end of line.

`int` An entry representing an integer value follows. Supported variables of the info type are: unit, range, description and output variable.

`float` An entry representing a float value follows. Supported variables of the info type are: unit, range, description and output variable.

`char` An entry representing a singel character follows. Supported variables of the info type are: description.

`string` An entry representing a string `string` follows. Supported variables of the info type are: unit and description.

`bool` An entry representing a boolean follows. Supported variables of the info type are: description.

`list` An entry representing a two way list of integers, floats or strings follows. Supported variables of the info type are: unit, range and description.

`array` An entry representing a array of integers or floats follows. Supported variables of the info type are: unit, range and description.

`table` An entry representing a table (two way list of a list) containing columns of integers, floats and/or strings follows. Supported variables of the info type are: description.

`result` An entry representing a result variable (*SiSi* writes result files automatically!).

Info Type (optional keywords): Optional keywords are available for different entries.

`\r`: The range of the entry. It only informs the user, in which range the value should be. Later a range check will be added.

`\u`: This entry only informs the user, of which unit the value is.

`\d`: Gives the user a description of the meaning of this parameter.

`\o`: If given, the entry will be handled as output variable (only for developpers!).

Note: Every parameter must have an individual name for correct identification!
Do NOT use reserved names!!!

Note: Do not use other control characters than newline an tab. Also do not use non ascii characters (e.g. language specific characters). Otherwise the parser could get confused.

Note: *SiSi* ignores any multiple white spaces (space, tab, return, ...)! Only whitespaces encapsulated in "..." will be considered. You can bring the file in any form you want. But after saving under **X***SiSi*, your layout and all comments beginning with # will be lost :-(.

marks the beginning of comments ignoring by the *SiSi* fileparser. Comments end with the end of line. If you write a parameter file with **X***SiSi* after reading it, all these comments will be deleted!!!

1.1.2 Example parameter files

A simple parameter file:

```
int     N_0     1
int     C       100
float   r       0.5
string  message "It's to late for sleeping."
```

An example parameter file containing all supported types:

```
# Parameter file written by XSiSi 2.0
# Last modified at Sun Apr 12 01:06:16 PDT 1998
comment test datei
int     N_0     1
        \u Individuals
        \r 0:
        \d Size of population at the beginnig of simulation.
float   r       0.5
        \u 1/year
        \d Population increase.
char    Version 'A'
        \d Version of photosynthese model.
string  loggingfile     "c:\home\formix\reference\logging4.dat"
boolean Test    true
        \d A test boolean variable.
array   TransitionMatrix
        \u Transitions/year
        \d Transitionmatrix from one habitattype to another.
        typeOfArray     float
        dimension       2 3
        head            "Spalte 1"
                        "Spalte 2"
                        "Spalte 3"
data
        1.1     1.2     1.3
        2.1     2.2     2.3
end
array   OneDimensional
        \d Test of an one dimensional array.
        typeOfArray     float
        dimension       3
data
        1.0     2.0     3.0
end
array   PlotInitialization
```



```
        \u Individuals
        \d This is the initialization of the plots.
        typeOfArray     float
        dimension       2 3
        file            "logging.dat"
table   Results
        column  "Time"  type    float
                \u years
                \d This is the actual simulation time.
        column  "x"     type    float
        column  "y"     type    float
        column  "file"  type    string
data
        1.0     0.0     100.0   "logging1.data"
        2.0     2.0     98.0    "logging2.data"
        3.0     5.0     95.0    "logging3.data"
end
list    Test
        type    array
                \u Transitions/year
                \d Transitionmatrix from one habitattype to another.
                typeOfArray     float
                dimension       2 3
                head            "Spalte 1"
                                "Spalte 2"
                                "Spalte 3"
data
        1.1     1.2     1.3
        2.1     2.2     2.3

        1.1     1.2     1.3
        2.1     2.2     2.3

        1.1     1.2     1.3
        2.1     2.2     2.3
end
```

1.1.3 Reserved parameter names

The following variables are used by the simulation programs:

`Time`	is the actual simulated time.
`TimeStart`	is the beginning of the simulated time.
`TimeEnd`	is the ending of the simulated time.
`TimeStep`	is the step of the simulated time.
`OutputStep`	is the time step of the output.
`RandomInit`	is the initialization value of the random generator.

Note: Please do NOT use reserved parameter names!!!

1.1.4 The syntax of a *SiSi* simulation file

A simulation file contains informations needed by every simulation like the model name and path, simulation name, unit, timestep, begin and end of simulation time, timestep of output, initialization of the random generator, and the list of all input files with parameters, state variables, scenarios, ... needed by the simulation.

1.1.5 Example simulation file

A simulation file:

```
# Simulation file written by XSiSi 2.0
# Last modified at Fri Apr 10 16:09:36 GMT+03:30 1998

string  SimulationName  "Referenzlauf"
                        \d (Only for information!)

string  ModelName       "Preymod 1.0"
                        \d (Only for information!)
string  ModelPath       "/usr/local/bin/preymod1.0"
                        \d The path of the executable model.

float   Time            0.0     \d Actual simulated Time.
                                \u years
                                \o # This is an output variable.
float   TimeStart       0.0     \d Start of simulation.
float   TimeEnd         100.0   \d End of simulation time.
float   TimeStep        0.1     \d TimeStep of simulation.
float   OutputStep      0.1     \d TimeStep of output (results).

int     RandomInit      0       \d Initializer for a random generator.

list    IncludeFiles            \d List of all included files.
        type    string
data
        "Parameters.par"
        "InitialState.sta"
        "Scenario.sce"
end

list    OutputVariables
        type    parameter
data
        result  Time
                type    float
                \u years
                \d Actual simulated Time.
                active  true
        result  x
                type    float
                \u Beute
                \d Beute
                active  true
        result  y
                type    float
                \u Raeuber
                \d Raeuber
                active  true
end
```


1.2 File Suffixes

`*.sim` Simulation files (see 1.1.4 on the preceding page).

`*.res` Result files.

`*.out` Outputs from simulation run.

`*.err` Errors from simulation run.

2 X*SiSi*

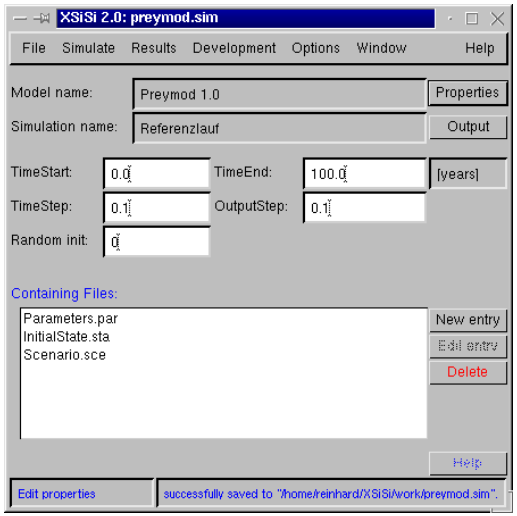

Figure 1: Top window of X*SiSi* (Screenshot under **Linux** with **XFree86**™ and KDE).

2.1 Main Window

Note: Please read the online documentation while testing X*SiSi* (Help buttons) for understanding how *SiSi* works!

In the main window you can manipulate the simulation files (time settings, ...). (See 1.1.4 on page 5)

2.2 Edit Options

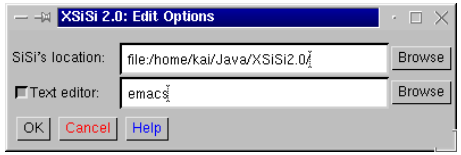

Figure 2: Window for editing options (Screenshot under **Linux** with **XFree86**™ and KDE).

***SiSi*'s Location:** This is the location of X*SiSi*.

Note: Don't forget to save these definitions with "Save Options" if you want to get these definitions as default at further times!

Note: X*SiSi* stores its options in a file called `"~\xsisi.ini"` (Windows) or `"~/.xsisi"` (Unix).

The meaning of `~` is platform dependent:

Unix: `~` means as usual your personal home directory (e.g. `/home/reinhard`).

Windows: `~` means under Windows the home directory of the java environment.

Note: Under Unix every user has its own optionfile (`"~/.xsisi"`).
Under Windows only one global optionfile exists (`"~\xsisi.ini"`).

X*SiSi* also creates the files `~/.xsisi_history` and `~/.xsisi_jobs` or rather `~\xsisi.his` and `~\xsisi.job` under Windows.

2.3 Edit Parameter

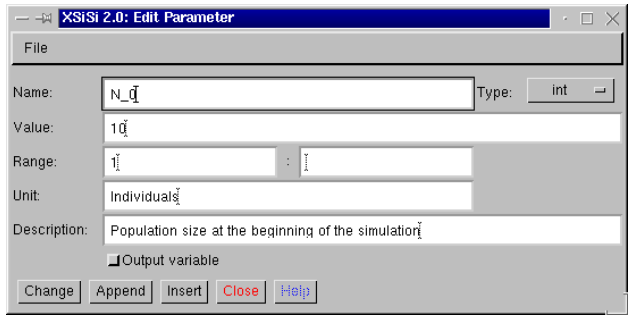

Figure 3: Window for editing singular parameters. (Screenshot under **Linux** with **XFree86**™ and KDE)

Name: The name of the parameter.

Type: Type of the parameter: `int`, `float`, `character`, `string`, `boolean` or `comment`. The other types `array`, `table` and `list` have extra windows for editing.

Value: The value of the parameter.
`int` : An integer
`float` : A float value
`character`: A single character
`string` : A string
`boolean` : A boolean value (true, false)
`comment` : A comment

Range: The range of the parameter. It isn't editable for all types. It only informs the user, in which range the value should be. Later there should be a value check integrated.

Unit: It only informs the user, which unit the value has. It isn't available for all types.

Description: Gives the user a description of the meaning of this parameter. It isn't available for type `comment`.

OutputVariable: Specify, if the parameter is an output variable. It isn't available for all types.

Change If a parameter with this Name already exists, you can change its settings.

Append Appends the parameter to the existing list.

Insert Insert the parameter in the existing list BEFORE the selected item.

Close Closes this window.

Help This help page will be shown.

2.4 Emit C++ Code

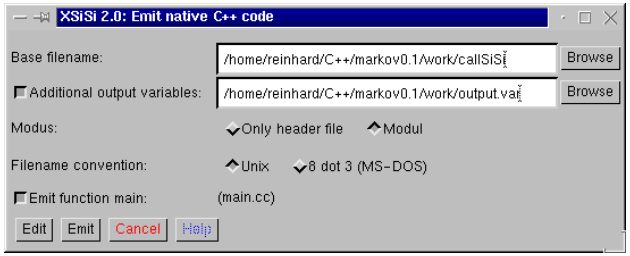

Figure 4: Emit C++ Code Window. (Screenshot under **Linux** with **XFree86**™ and KDE)

If you have created a simulation file (see 1.1.4 on page 5) and the input files and you want to include these files in your own simulation environment, than create with the menu entry "Emit C++ Code" a header file (`*.hh`), which you have to include in your source code. Please read first 3.1 on page 11!

Base filename: File without suffix to create. **X***SiSi* will create *(base)*`.hh`, `mysisi.hh` and if the *modus modul* is chosen *(base)*`.cc`.

Additional output variables: In this file you can declare additional output variables without any initialization, which will be declared in *(base)*`.hh`. The user can only select these variables for output but cannot set their values.

Modus: Choose *Only header file* if you include *(base)*`.hh` only one time in your source code. But if you have different modules with different object files including *(base)*`.hh` you have to generate a modul. **X***SiSi* will create additional a file *(base)*`.cc` and declares all variables in *(base)*`.hh` as external.

Filename convention: If *8 dot 3 (MS-DOS)* is selected, then **X***SiSi* will generate files using DOS*SiSi* (see 4 on page 15) and using class `SiSi_String` instead of `String`. Use this option, if you platform doesn't support long filenames or your compiler has already declared the class `String`.

Emit function main: This feature is only for testing correct reading of the *SiSi* file parser. Use this only, if you want to check correct reading of your simulation parameters.

Edit Edit the output variables.

Emit Emits the desired code.

3 Simulation development

3.1 The easy way to include the *SiSi* fileparser

Imagine you have written a simulation environment like this, which you call e.g. by the main function:

```
void simulation()                    // A simple example:
{
  double a = 1;                      // spez. Wachstumsrate der Beute.
  double b = 1;                      // spez. Beuteverlustrate der Beute.
  double c = 1;                      // spez. Beutegewinnrate des Raeubers.
  double d = 1;                      // spez. Atmungsrate des Raeubers.
  double k = 2;                      // Tragfaehigkeit fuer Beutepopulation.
  double x = 0.1;                    // Beutepopulation.
  double y = 0.1;                    // Raeuberpopulation.

  for( double time=0; time<20; time+=0.002 ) {
    x += (a*x*(1-x/k) - b*x*y) * 0.002;
    y += (c*x*y - d*y) * 0.002;
  }
}
```

The great disadvantage of this solution is, that you have to recompile this code if you want to vary some parameters. So it's better to read the parameters `N_0`, `r` and `C` from a file.
Create a simulation file named e.g. `"preymod.sim"` understood by *SiSi* like this. You can do that with your favourite editor or with **X***SiSi*:

```
string  SimulationName  "Reference run"
string  ModelName       "Preymod 1.0"
string  ModelPath       "/usr/local/bin/preymod1.0"
float   Time            0.0     \d Actual simulated Time.
float   TimeStart       0.0     \d Start of simulation.
float   TimeEnd         20.0    \d End of simulation time.
float   TimeStep        0.002   \d TimeStep of simulation.
float   OutputStep      0.1     \d TimeStep of output (results).
int     RandomInit      0       \d Initializer for a random generator.
list    IncludeFiles            \d List of all included files.
        type    string
data
        "preymod.par"
end
list    OutputVariables
        type            parameter
data
        # No output.
end
```

Create a parameter file named e.g. `"preymod.par"`:

```
float a         1
float b         1
float c         1
float d         1
float k         2
```



```
float x         0.1
float y         0.1
```

Now you have to generate a header file named e.g. `"preymod.hh"`. Do this with **X***SiSi* (menu entry: "Emit C++ Code") (see 2.4 on page 9) and modify your simulation programm like this:

```
#include "preymod.hh"                  // Include automatically generated code.
// Code generated by xsisi menu item Development->Emit C++-Code.
// (All parameters are declared as global and are now available!)

void simulation()               // A simple example:
{
  for( Time=TimeStart; Time<TimeEnd; Time+=TimeStep ) {
    // The model equations:
    x += (a*x*(1-x/k) - b*x*y) * TimeStep;
    y += (c*x*y - d*y) * TimeStep;
  }
}

int main ()
{
  if( !SiSi::parseSimulation("preymod.sim", "preymod") ) // <------- !!!
    return 0;                                            // Error!!!
  simulation();
  return 0;
}
```

That's it. Now, you can manipulate the parameter file `"preymod.par"` and start the simulation without recompiling!

Summary: You have to include only 2 (TWO) lines in your code:

```
#include "preymod.hh"
SiSi::parseSimulation("preymod.sim", "preymod");
```

Note: You have to link the sisi.lib (Windows) or the libsisi.a (Unix) to your Programm.
For Compiler specific instructions see: 3.3 on the facing page

3.2 Complete support by X*SiSi*

If you want to

- manipulate parameters with **X***SiSi*,
- start simulation from **X***SiSi*,
- write result files understanding by **X***SiSi*,
- want to produce graphical outputs with **X***SiSi* and
- control the progress of simulation runs with **X***SiSi* job controlling system,

you have to modify your source code like this:

```
#include "preymod.hh"                // Include automatically generated code.
// Code generated by xsisi menu item Development->Emit C++-Code.
// (All parameters are declared as global and are now available!)

void simulation();

int main (int argc, char* argv[])
{
  if( !SiSi::parseSimulation(argc, argv) ) // Read all parameters. <------- !!!
    return 0;                              // Error!!!
  if( TimeStep <= 0 ) {
    MessageHandler::error("Timestep must be greater than zero!");
    SiSi::abort();                         // Abort simulation.
    return 0;
  }
  MessageHandler::information("This is preymod 1.0");
  MessageHandler::information("--------------------");
  MessageHandler::information((String) "Using simulation " +
                              SiSi::SimulationName);
  SiSi::resultWriter.open();               // Opens default result file.
  simulation();
  SiSi::finalize();                        // Closes all files and finishs ...
  return 0;
}

void simulation()
{
  Total = x + y;
  SiSi::resultWriter.output(TimeStart, true);
    // Write initial state ignoring setTimeStep()!
  for( Time=TimeStart; Time<TimeEnd; Time+=TimeStep ) {
    SiSi::resultWriter.output(Time);       // Write state to resultfile.
    SiSi::logFile.progress(Time, TimeEnd); // Write progress to logfile.

    // The model equations:
    x += (a*x*(1-x/k) - b*x*y) * TimeStep;
    y += (c*x*y - d*y) * TimeStep;
    Total = x + y;
  }
  SiSi::resultWriter.output(TimeEnd, true);// Write final state to resultfile.
}
```

3.3 Different Compilers

3.3.1 Gnu C++ Compiler

Here you can find everything to develop a simulation environment supported by **X***SiSi* with the Gnu's C++ Compiler.

1. Please produce your simulation file and parameter files (see 3.1 on page 11).
2. Please produce a headerfile with (see 2.4 on page 9).
3. Please change your source code (see 3.1 on page 11).

4. No you have a headerfile called e.g. `preymod.hh` and your source-code called e.g. `preymod.cc`.
5. Please type in the directory where your sources lie:

```
gcc  -I/home/reinhard/C++/sisi2.0/include -L/home/reinhard/sisi2.0/lib \
     -o preymod preymod.cc -lsisi2.0
```

`-I/home/reinhard/C++/sisi2.0/include`	specify the path of *SiSi*'s include files.
`-L/home/reinhard/sisi2.0/lib`	specify the path of *SiSi*'s libraries.
`-o preymod`	specify the name of the binary.
`-lsisi2.0`	links the *SiSi*-library.)

6. Please copy your binary (e.g. `preymod`) to the directory you specified in `preymod.sim`.
7. Ready. Now **X***SiSi* supports your simulation environment.

Note: It's mostly easier to create a simple makefile. Then you have only to type "make".

3.3.2 Borland C++ Compiler

Here you can find everything to develop a simulation environment supported by **X***SiSi* with the Borland C++ Compiler.

1. Please produce your simulation file and parameter files (see 3.1 on page 11).
2. Please produce a headerfile with (see 2.4 on page 9).
3. Please change your source code (see 3.1 on page 11).
4. No you have a headerfile called e.g. `preymod.hh` and your source-code called e.g. `preymod.cc`.
5. Now you have to build a project under Borland C++ Development environment (this example is for a german version):

Projektname	: `preymod (Anwendung [.exe])`
Umgebung	: Win32
Zielmodell	: Console
Standardbibliotheken	: Klassenbibliothek, Laufzeitbibliothek

Projektdateien	`-preymod [.exe]`
	`libSiSi [.lib]`
	`preymod [.cc]`

Verzeichnisse:	Include: `c:\bc4\include;c:\SiSi2.0\include` Lib : `c:\bc4\lib;c:\SiSi2.0\lib`

(So, you have to specify the path of the Compiler's files and the path of *SiSi* e.g. `"c:\SiSi2.0"`.)

6. Compile this Projekt (it should produce e.g. `preymod.exe`).
7. Please copy your binary (e.g. `preymod`) to the directory you specified in `preymod.sim`.
8. Ready. Now **X***SiSi* supports your simulation environment.

4 FAQ

This section shows a list of frequently asked questions.

1. **What is the difference between DOS*SiSi* and *SiSi*?**
   DOS*SiSi* is automatically generated from *SiSi* by an unix shell script, which renames all long filenames by short filenames supported by e.g. MS-DOS and also substitutes all `#include` lines. Additional all occurence of `String` are replaced by `SiSi_String` for avoiding conflicts with compilers declaring type or class `String`. Use DOS*SiSi* with e.g. older Borland compilers (before Version 5.0).

2. **I want to use variables in structs or as member variables in classes. Does *SiSi* support such member variables like e.g. `Plot->Tree`, `Plot.Tree` or `Plot::Tree`?**
   Yes. If you give such variables names in the input files (or under **X***SiSi*), which aren't single identifiers[1] then *SiSi* don't declare them in `callSiSi.hh` creating with *Emit Cxx Code* (see 2.4 on page 9). But *SiSi* parses these variables from the given files. So *SiSi* needs the declaration of these variables in `MySiSi.hh`. Please insert in `MySiSi.hh` the declaration of the structs or classes for these variables or insert in `MySiSi.hh` `#include` lines including the desired header files.

3. **If X*SiSi* do one action after calling it for the first time, I have to wait (e.g. for the first question dialog). What's happened?**
   X*SiSi* is a 100% pure Java application running on a virtual machine (e.g. JRE). (The developers of **X***SiSi* have compiled the source code in a Java binary code, understood by such a virtual machine). If you call a part of **X***SiSi* for the first time, it will be compiled by the virtual machine just in time. So you have to wait for compilation. After finishing your **X***SiSi* session, all compiled code will be forgetten :-(The advantage of this virtual machines is, that **X***SiSi* will be supported by *all* platforms running a Java virtual machine. There are also compilers on the different platforms available which compiles the code complete to machines binary code. So you can call this binary without any virtual machines but only on one platform. The disadvantage is, that there is no free compiler available :-([2]

[1] Identifiers consist of letters, digits (not at first position) and/or underscores.
[2] If you have a free compiler, please contact one of the authors!

5 Availability

5.1 ftp server

The actual *SiSi*-Distribution is available under

ftp://ftp.usf.uni-kassel.de/pub/sisi/

If there are any `README` files, please read them first!

5.2 *SiSi*'s home page

You can find a link to *SiSi*'s actual home page here:

http://www.usf.uni-kassel.de/~reinhard/

6 Copyright ©1997, 1998

[3] You find a copy along this program: Please read the file `LICENSE` in the top *SiSi* directory.

Index

#, 3

Availability, 16

bool, 2
Borland C++ Compiler, 14

char, 2
comment, 2
Copyright, 16

DOS*SiSi*, 9, 15

FAQ, 15
file suffixes, 6
ftp server, 16

Gnu C++ Compiler, 13
GNU General Public License, 16

Html, *see* *SiSi*'s home page

info type
    description, 2
    range, 2
    unit, 2
input files, 2
    syntax, 2
int, 2
Internet, *see* *SiSi*'s home page

Java, 15

`OutputStep`, 4

parameter files
    examples, 3

`RandomInit`, 4
reserved parameter names, 4

simulation
    development, 11
simulation development, 11
simulation files
    example, 5
    syntax, 5
SiSi's home page, 16
string, 2

`Time`, 4
`TimeEnd`, 4
`TimeStart`, 4
`TimeStep`, 4

WWW, *see* *SiSi*'s home page

X*SiSi*, 7
    Complete Support, 12
    Edit Parameter, 8
    Emit C++ Code, 9
    main window, 7
    Options, 7
    SiSi's location, 7
`.xsisi`, 8
`xsisi.ini`, 8