

FROGS

Event-driven Simulator

Version 1.2

User Manual

July 2001

Contents

1.Introduction.....	iii
1.1Overview.....	iii
2.Generating a simulation program.....	iv
2.1Description of a simulation program.....	iv
2.2The configuration classes.....	iv
2.3A typical simulation Makefile.....	iv
3.Configuring the Simulation.....	vii
3.1The Librarian.....	vii
3.1.1Simulation Models Concept.....	viii
3.1.2Architecture Concept.....	viii
3.1.3Typical Architecture.....	ix
3.1.4Loading Simulation Modules.....	ix
3.1.5Creating/Copying a Model Instance.....	ix
3.2Modifying/Renaming a Model Instance's Configuration.....	x
3.3Destroying a Model Instance.....	xi
3.4Defining an Architecture.....	xi
3.4.1Associating Model Instances.....	xi
4.Simulation Project.....	xiii
4.1Creating a Project.....	xiii
4.2Modifying a Project.....	xiv
4.3Selecting a Project.....	xiv
5.Configuring an Architecture.....	xv
5.1Configuring Events.....	xv
5.1.1Events and the Simulation Scenario.....	xv
5.1.2Event Source Settings Syntax.....	xv
5.1.3Generating Events Automatically.....	xv
5.1.4Generating Events Manually.....	xvi
5.2General Simulation Settings.....	xvii
5.3Simulation Tool Parameters.....	xviii
5.4Operating Options.....	xix
6.Exporting/Importing the work environment.....	xxi
6.1Exporting/Importing the current project.....	xxi
6.2Exporting/Importing the models library.....	xxi
7.Running a Simulation.....	xxii

7.1 Interactive Execution.....	xxii
7.1.1 Starting Simulation.....	xxii
7.1.2 Controlling Simulation.....	xxiii
7.1.3 Simulator Status Indicator.....	xxiv
7.1.4 Using the Inspector.....	xxiv
7.1.5 Placing Manual Requests.....	xxiv
7.1.6 Setting Programmed Awakening Functions.....	xxv
8. Displaying Statistics Graphs.....	xxvii
8.1 Selecting Graphs.....	xxvii
8.2 Display by Type of Graph.....	xxviii
8.2.1 Time Curves.....	xxviii
8.2.2 Composite Curves.....	xxviii
8.2.3 Histograms.....	xxix
8.3 Controlling the Simulation.....	xxix
8.3.1 Continuing/Stopping Simulation.....	xxix
8.3.2 Setting Breakpoints.....	xxix
8.3.3 Displaying and Checking Breakpoints.....	xxix
8.4 Scale Compression.....	xxx
8.4.1 Y Compression.....	xxx
8.4.2 X Compression.....	xxx
8.4.3 Zoom In.....	xxxi
8.4.4 Zoom Out.....	xxxi
8.5 Composite Curves.....	xxxi
8.6 Placing Graphs.....	xxxi
8.7 Selection and Cross Hairs.....	xxxi
8.7.1 Actions on Graph Sections.....	xxxi
8.7.2 Using the Cross Hairs.....	xxxii
8.8 Other Local Functions.....	xxxii
8.8.1 Local Time Curve Functions.....	xxxiii
8.8.2 Local Histogram Functions.....	xxxiii
8.8.3 Common Functions.....	xxxiv
8.9 General Options.....	xxxv
8.9.1 Adjusting Abscissas.....	xxxv
8.9.2 Scroll Lock.....	xxxv
8.9.3 Auto-Select Color.....	xxxv
8.9.4 Auto-Save Session.....	xxxv
9. Command Lines.....	xxxvi
9.1 ISE Start Options.....	xxxvi
9.2 Simulator Start Options.....	xxxvi

1. Introduction

1.1 Overview

FROGS is an object-oriented, general-purpose simulation tool designed for efficient and fast behavioral evaluation of complex architectures. It can be used interactively either as a tool for aiding design and tuning of time-constrained systems or as a decision aid tool for adjusting the behavior of these systems in production.

FROGS exhibits event-driven simulation techniques based on a unique, internally maintained time reference, which is totally independent from the host workstation's idea of time.

FROGS consists of :

- ⑩ SIMEX which is a simulation kernel providing a rich set of C++ classes usable for system modelling, such as threads or activities, synchronization objects, measurement tools and statistical support.
- ⑩ A GUI application for interactive configuration and on-line monitoring of the simulation. This application provides a graphical integrated simulation environment (aka the **ISE**) aimed at controlling the simulation process. For instance, the ISE allows you to display and analyze selected parameter evolution, such as state transitions of simulation objects.

This document details the way simulation executables can be built, and how to use the FROGS' ISE.

2. Generating a simulation program

2.1 Description of a simulation program

Using FROGS means executing simulation program that exhibits a given behavior you need to evaluate. Each simulation is made of a configuration stage where the simulated architecture and its components are defined, and their properties or parameters tuned. Then comes the execution stage where you can observe the simulation behavior using a GUI monitor.

A simulation program consists of :

- ⑩ A model implementation library. This is a shared library logically divided in two distinct functional parts. The first part is aimed at providing the *model's configuration classes* the FROGS' GUI will dynamically use to display a graphical configuration interface for the external attributes of your model. The second part is the simulation code itself, mainly based on the SIMEX library, which describes the model behavior.
- ⑩ A native executable, linked to your model library and to the FROGS' system libraries. This is usually a small program aimed at bootstrapping the simulation, using the appropriate FROGS' executive entry–point.

2.2 The configuration classes

The FROGS' librarian allows you to access and update the defined attribute values through a Tcl/Tk–based GUI. This way, you are able to change your simulation settings for a given configuration without having to rebuild your simulator.

These attributes are gathered into *configuration classes*. The configuration class abstraction is implemented through specialized C++ classes that provide dynamic support to the GUI for displaying automatically–generated configuration windows containing the model's parameters. The collected values are finally stored into a configuration database.

When the simulator starts, the simulation system loads the configuration database maintained by the FROGS' librarian, and activates the configuration objects according to the selected simulation architecture, which in turn are expected to instantiate the simulation objects required to populate the “simulated world”.

The tunable attributes of your simulation must be defined in the model library. The *ExConfig.cc* example file gives a detailed explanation of this process.

The configuration database support code is based on the **cfclass** package.

2.3 A typical simulation Makefile

The *examples/cfsim* directory contains an example that illustrates how to implement configuration classes, the C++ support one should provide in a shared library to use FROGS' GUI–based simulation configuration and execution interfaces. The contributed *Makefile* is Linux–based, but should be easily ported to other POSIX environments. We will use this *Makefile* to illustrate the process of building a simulation program.

The following *Makefile* builds a shared library – namely *libcfsim.so* – and the simulation executable named *cfsim*. *libcfsim.so* contains the simulation model (*ExModel.cc*) and the configuration support code (*ExConfig.cc*). The *Main.cc* source file is aimed at bootstrapping the simulation process. See 9.2, Simulator Start Options to determine

which options can be passed to start the simulator from the command line instead of using the FROGS' ISE. Usually, you should only have to give the `-C <archName>` option.

The configuration support code needs to be parsed by the *obscan* tool, which is part of the **obstick** package. After *ExConfig.cc* is parsed, a C++ source file – namely *DbImpl.cc* in our example – is produced, containing methods to support the persistence of instances from the *ExConfigA* and *ExConfigB* classes. Each instance holds a list of persistent configuration attributes which describes its settings. The memory layouts of the various persistent classes found while parsing *ExConfig.cc* are determined then saved in a repository named *DbImpl.pcr*.

Note that `-Wl,-export-dynamic` must be passed to the *GNU* linker as some FROGS internals attempt to resolve C/C++ routine addresses at run-time by scanning the current executable's namelist.

Ensure your **LD_LIBRARY_PATH** variable is updated to point to the directory you are compiling this example in before running the simulation, or set the `-rpath` directive from the *Makefile* accordingly.

Here is the example Makefile:

```
CC = c++
RM = rm -f
LN_S = ln -s
# Set the FROGS variable to point to your installation root
FROGS = /usr/local/frogs

OBSCAN = $(FROGS)/bin/obscan
INCLUDES = -I. -I$(FROGS)/include
# -fwritable-strings is for Tcl-related code
CXXFLAGS = -fwritable-strings -fpcc-struct-return -fnonnull-objects -fno-exceptions
LIBS = -L$(FROGS)/lib -lsimex -lstatobj -lcfclass -lobstick -ldevkit -ltoolshop -ltcl -lelf -lm -lnsl -ldl

all: cfsim

# "cfsim" is the executable used to bootstrap the simulation
cfsim: libcfsim.so Main.o
    $(CC) -o $@ Main.o -L. -lcfsim $(LIBS) -Wl,--export-dynamic -Wl,-rpath -Wl,$(FROGS)/lib

# ExModel.cc contains a dumb simulation model
ExModel.o: ExModel.cc
    $(CC) $(CXXFLAGS) -c -g $< -o $@ $(INCLUDES)

# The main routine containing the bootstrap code is in Main.cc
Main.o: Main.cc
    $(CC) $(CXXFLAGS) -c -g $< -o $@ $(INCLUDES)

# This is how to build the model library
libcfsim.so: DbImpl.o ExConfig.o ExModel.o
    $(RM) libcfsim.so*
    $(CC) -shared -o libcfsim.so.0.0.0 DbImpl.o ExConfig.o ExModel.o -Wl,-soname -Wl,libcfsim.so.0
    $(LN_S) libcfsim.so.0.0.0 libcfsim.so
    $(LN_S) libcfsim.so.0.0.0 libcfsim.so.0

# ExConfig.cc defines two configuration classes for our simulation model
```

```

ExConfig.o: ExConfig.cc
    $(CC) $(CXXFLAGS) $(INCLUDES) -fPIC -c -g -o $@ $<

# Update the p-class repository with the layout information from
ExConfig.cc
DbImpl.pcx: ExConfig.cc
    $(OBSCAN) -i ExConfig.cc -r $@ $(INCLUDES)

# Generate the obstick support code from the p-class repository contents
DbImpl.cc: DbImpl.pcx
    $(OBSCAN) -o $@ -r $<

# The compiled obstick support code is part of the model library
DbImpl.o: DbImpl.cc
    $(CC) $(CXXFLAGS) $(INCLUDES) -fPIC -c -g -o $@ $<

ExModel.cc Main.cc: ExModel.h

ExConfig.cc: ExConfig.h

clean:
    rm -f *.o *.pcx cfsim libcfsim.so* DbImpl.cc *~

.PHONY: clean

```

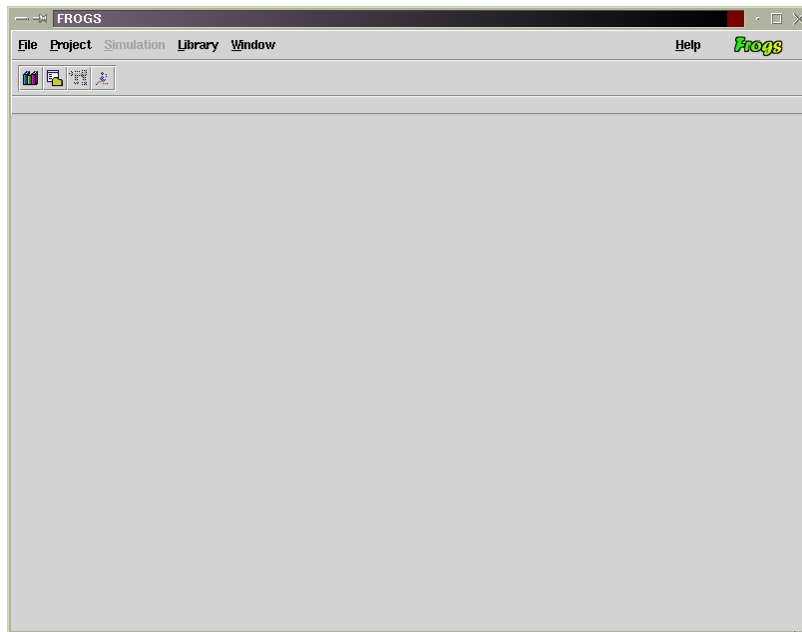
3. Configuring the Simulation

The second of the two stages needed to set up a simulation consists of defining the simulated architecture. This step is controlled by a central graphic application combining all simulation session configuration and operation services.

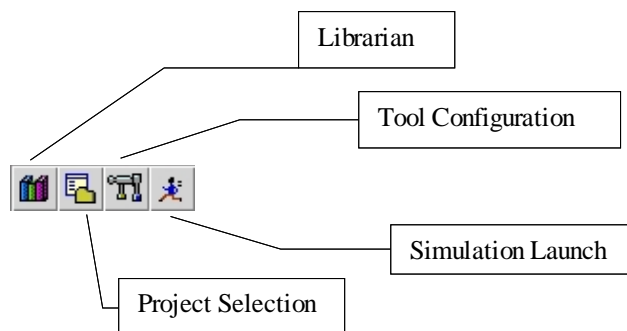
To run the central graphic interface, run the following command:

```
$ frogs&
```

The ISE originally appears as follows:

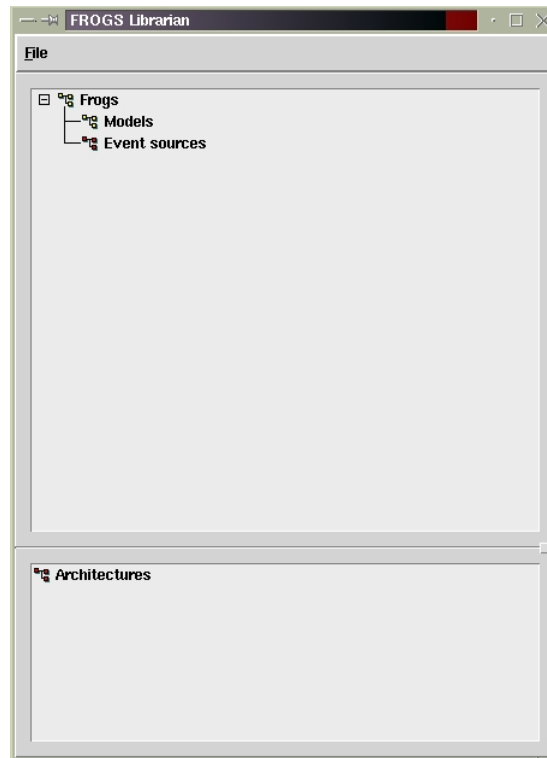


The main toolbar's functions are as follows:



3.1 The Librarian

The Librarian is used to define the simulated architectures and their respective components. The components' parameters can be tuned using this GUI. In the ISE, select the **Open** command from the **Library** menu, or click the icon (to the left of the toolbar). The following screen is then displayed.



This window shows the known models hierarchy, as well as the different existing instances. Two separate directory structures can be seen:

- ⑩ The first, at the root of which is the **Frogs** element, organizes models according to whether they belong to the simulated models (**Models**), or event sources (**Events**) class. In our example, we will define an instance of the **Islan** model, as well as an event source.

All instances of the **Events** model must be associated with the model's configuration via the **Event sources** resource, which can be accessed from the configuration's **Events** tab.

- ⑩ The second, at the root of which is the **Architectures** element, lists the existing simulation configurations as a single level.

3.1.1 Simulation Models Concept

FROGS includes an object-oriented simulation executive offering services for supporting models written in C++. Each simulation model describes the behavior of a given activity, to an accuracy selected by its designer.

3.1.2 Architecture Concept

Once the various configurations of useful models have been individually defined, they must be associated together before the actual simulation can begin. The result of associating them together is called an *architecture*. An architecture is a high level description associating models together in their respective configurations according to a given hierarchy.

An architecture may be designed as a super-model, whose role is to associate the various elements of a simulation system to define its initialization rules. That process consists of three main phases:

- ⑩ Creating the object simulations representing the simulated world (for example, creating our Islan example's instance; see below);
- ⑩ Optionally creating event sources which are aimed at triggering actions on a timely (or manual) basis;
- ⑩ Setting up event generators.

3.1.3 Typical Architecture

In its simplest form, a FROGS simulation consists of an architecture comprising one model, of which there is one instance.

All information on the different known architectures is stored in a library file containing the simulation models. Only one library is active at any time within ISE, and is only updated via the Librarian. The configuration database and the models library are the same file.

3.1.4 Loading Simulation Modules

Simulation models are contained in shared libraries on the host system.

These models must be explicitly loaded by the Librarian before they can be used. These model libraries export one or more definitions of simulation models, along with their configuration characteristics. The Librarian makes use of all that information in order to offer a suitable graphic configuration interface.

The FROGS Librarian's role is therefore to offer a consistent graphic interface for creating and configuring the chosen instances according to the required characteristics of behavior.

The various configurations of models are organized within a library of simulated components. Using this approach, they can be reused in different contexts simply by association.

Simulation models can be preloaded when the Librarian is first run under a given user account, using an automatic loading system triggered when an **autoload** file is found in the **share/frogs** subdirectory of the installation's root directory.

This text file lists certain external resources that can be attached dynamically to the ISE. Each model configuration module begins with the **cfmod=** keyword followed by the library file's generic name. For example, the Islan configuration module is loaded by the expression **cfmod=islant**. The corresponding module file can be accessed from the installation's root directory, in the area reserved for libraries (e.g. lib/libislant.so on Linux platforms).

3.1.5 Creating/Copying a Model Instance

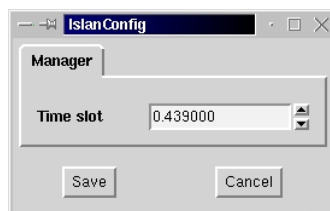
- ⑩ To create our example's Islan instance, right-click the **Islant** node in the models directory structure, and then select the **New** command to display the following dialog box:



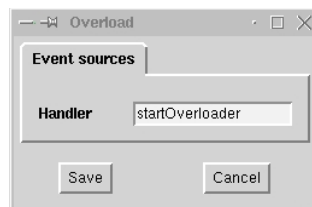
- ⑩ Type the new instance's name, for example **IslanConfig**, and then validate with the **ENTER** key or click the **OK** button. After each new instance is created, the librarian automatically edits the new instance.
- ⑩ To duplicate an existing configuration, select the **Copy** command from the previous menu. A dialog box is then displayed, in which you enter the copy's name.

3.2 Modifying/Renaming a Model Instance's Configuration

- ⑩ Each instance of a simulation model exports a series of settings through which it can be configured. When an instance is created, a default set of values chosen by the model's designer is initialized. If you wish to modify those settings, double-click the selected instance's name, or right-click the sheet representing it to display the contextual menu from which you select the **Open** command. The **IslanConfig** configuration has only a single parameter, which is the simulated network's time slot duration. If you use this command on the **IslanConfig** instance that you have just created, the following configuration window is displayed:



- ⑩ Each tab represents a specific category of the model's configuration. For the pre-defined event source model, assuming that you've just created the **Overload** event instance, the settings are as follows :



The *startOverloader* routine used in the previous screenshot is implemented by the *Islan* simulation code (C binding required here). It is aimed at creating an overload situation for the *Islan* network. You should replace it by the function identifier which is adapted to your simulation code.

When all the settings have been correctly updated, validate the modifications with the **Save** button.

- ⑩ If you wish to rename a specific configuration, right-click the icon representing it in the models directory structure, and then select the **Rename** command in the contextual menu. A dialog box is then displayed, in which you enter the instance's new name.

Renaming an instance does not affect any existing links between the different instances.

3.3 Destroying a Model Instance

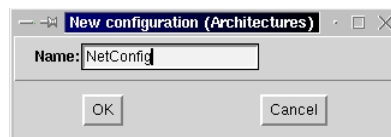
If you wish to delete a model configuration, right-click the icon representing it in the models directory structure, and then select the **Delete** command in the contextual menu. A dialog box is then displayed, asking you to confirm the action.

Deleting an instance automatically destroys links from other model configurations. That/those model(s) can therefore no longer work correctly if those links must exist.

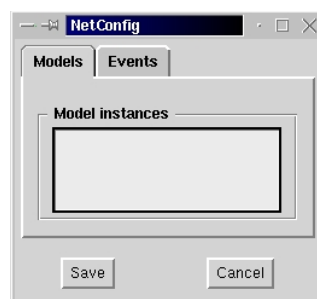
3.4 Defining an Architecture

3.4.1 Associating Model Instances

An architecture is an association of different simulation models that are intended to work together within the FROGS simulator. To create an architecture from the FROGS Librarian, right-click anywhere in the second directory structure called **Architectures**, and then select the **New** command in the contextual menu. A dialog box is then displayed, in which you enter the new architecture's name, e.g. **NetConfig**:

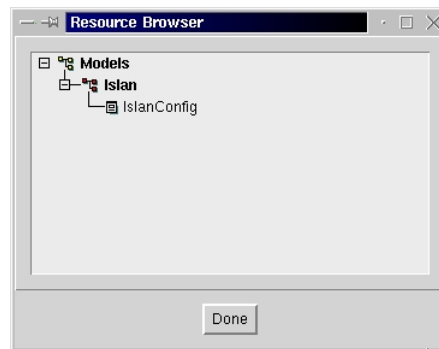


Validate with the **ENTER** key or click **OK** to continue creating the architecture. When it has been created, the new architecture is displayed so that its characteristics can be defined. The following window is displayed:



The graphic representation of an architecture displays two tabs: one for user-defined model instances (i.e. **Models**), and the other for event source instances (i.e. "Events"). The first phase consists of choosing the user-defined models. Right-click the simulation models list (i.e. **Models**), and then select the **Insert** command in the

contextual menu. This function is used to choose configurations of model instances that will be associated together in the simulation. In our example, the displayed selection window appears as follows:



Double-click **IslanConfig** to include the model instance already defined in the architecture. When you have done so, click the **Done** button.

You should go through the same process to attach the **Overload** event to the simulation, by first selecting the **Events** tabs.

4. Simulation Project

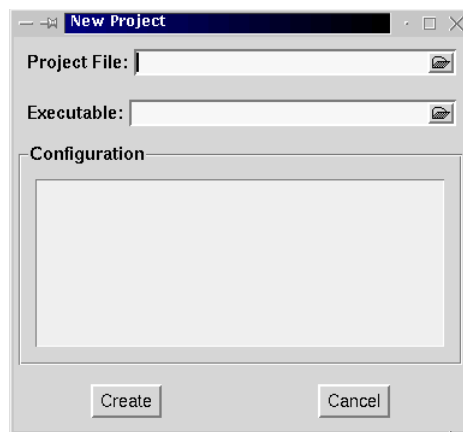
An FROGS *simulation project* is created by associating an architecture with a simulation executable.

All information regarding a specific project is stored in a separate file, which refer to the standard model library that was active when the file was created.

Before making the first simulation of the Islan example, that association must be made using the **Project** menu's commands.

4.1 Creating a Project

Open the **Project** menu, and then select the **New** command. A window is then displayed, offering two selection fields: one to select the new **Project File**, and the other to select the associated architecture (**Configuration**).



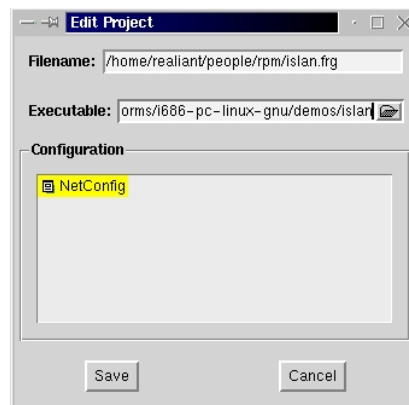
- ⑩ In the first field, enter the project file's access path. If you wish, you can use a browser by clicking the icon displayed to the right of the input field. The file must not already exist. FROGS automatically adds the **.frg** extension, which refers to all simulation projects' filename.
- ⑩ You can enter the simulation executable's name in the second field. Environment variables may be used to represent all or part of the access path, and will only be expanded when the ISE actually uses the access path. Note that the simulation executable's name need not be input at this point when the project's initial details are entered. In that case, the project must later be modified to complete its definition once the simulation executable is specified.
- ⑩ Select a simulation architecture in the displayed list, by clicking its name so that it is highlighted. Our example architecture **NetConfig** appears in this list.
- ⑩ After you have specified all the above information, click the **Create** button to create the project.

The new project is automatically selected by the ISE.

The ISE's commands for running a simulation can only be accessed when a current project has been selected and the project's configuration specifies a valid executable and simulation architecture.

4.2 Modifying a Project

After selecting the current project, select the **Edit** command from the **Project** menu. A window is then displayed, offering two input fields: one to select the simulation **Executable**, and the other to select the associated architecture (**Configuration**). The current project's filename appears at the top of the dialog box, in read-only form.



- ⑩ Enter the access path and name of the simulation's executable file in the field labeled **Executable**. If you wish, you can use a browser by clicking the icon displayed to the right of the input field. The selected file must be a simulation executable generated separately by the normal FROGS procedure. Until a valid executable file has been specified in this field, all the ISE's commands for running the simulation are inactive. This field may contain environment variables that will be evaluated when the access path is actually used.
- ⑩ Select a simulation architecture from the proposed list by clicking its name to highlight it. In our list, we will find our example architecture **NetConfig**.
- ⑩ When you have selected the previous two items, click the **Save** button to modify the project.

4.3 Selecting a Project

To change the ISE's current project, select the **Open** command from the **Project** menu. A browser is then displayed, with which you can select the chosen project file. Double-click the filename displayed in the browser, or select the file with the right mouse button and then click **OK**. From then on, all the ISE's commands will act upon the new active project, whose name is displayed in the main window's title bar.

The most recently opened projects can be quickly accessed using the **File** entry in the **Projects** menu.

5. Configuring an Architecture

Configuring an architecture consists both of choosing the laws governing how the different event sources defined for the architectures are generated, and making certain settings in the control tools or the simulator itself. All these settings are stored for the active project.

5.1 Configuring Events

5.1.1 Events and the Simulation Scenario

A simulation scenario is defined by choosing laws governing how the events defined for a given simulation configuration are generated. FROGS offers seven predefined generation laws, each of which defines how frequently the service routine defined for each event source is called. Configuring an event source consists of setting a selector and entering a generation law setting. The following summary lists the types of predefined event sources, with the graphic selector position and corresponding setting syntax for each. Both settings must be provided for each source appearing in the configuration's **Event sources** window.

If no event source is defined for the current simulation architecture, the **Event Sources** window is not displayed.

5.1.2 Event Source Settings Syntax

Selector	Type of call	Setting syntax
Periodical	Periodic	[t0–tmax/]<period>
Exponential	Exponential	[t0–tmax/]<mean>
Uniform	Uniform	[t0–tmax/]<dmin–dmax>
File	Source file	<source file>
Manual	Manual	<destination file>
Timer	Unique	<time (simulated time)>
Null	–no call–	–no setting–

5.1.3 Generating Events Automatically

An automatic source is handled by the simulator without any action by the user being needed.

An automatic source relates to a domain [t0–tmax] representing the simulated time frame during which the source is active. No event can be generated outside its limits. If no limits have been defined, the source is active throughout the simulation. A simulated time limit is expressed by an actual value, that may be associated with a unit of time, e.g. 100 msc (usc = microseconds, msc = milliseconds, sec = seconds). The default unit is the microsecond (usc). If the tmax limit's units are not specified, it always uses the unit chosen for limit t0. The limits are separated by a hyphen, and separated from the rest of the setting by one of the characters “/”, “:” or “.”. The following syntaxes can also be used:

Syntax	Meaning
tx/...	tx-<simulation end>
-tx/...	<simulation start>-tx
tx-/...	tx-<simulation end>

- ⑩ For periodical sources, the **period** setting is a duration representing the generation law period, expressed in simulated time. It is expressed in the same way as a simulated time limit.
- ⑩ For exponential sources, the **mean** setting is a duration representing the mean of the generation law, expressed in simulated time.
- ⑩ For uniform sources, the **dmin-dmax** setting is a domain expressed by two simulated-time limits representing the minimum and maximum limits of the generation law. If **dmax** is omitted, the domain used is equal to [0-2*dmin].
- ⑩ For file sources, the setting is the access path for the file containing generation start time instructions, with one start time in each line. So that the rest of the file is correctly interpreted, the first line must begin with the marker “# \$@timelog”. All other lines in the file are evaluated according to the same rules as a simulated time window limit, until an invalid character or end of line is reached (spaces and tabs are ignored). Any line beginning with the “#” character is seen as a comment; empty lines are permitted and ignored. For example, the following file extract lists three start times expressed in microseconds, followed by two others expressed in seconds.

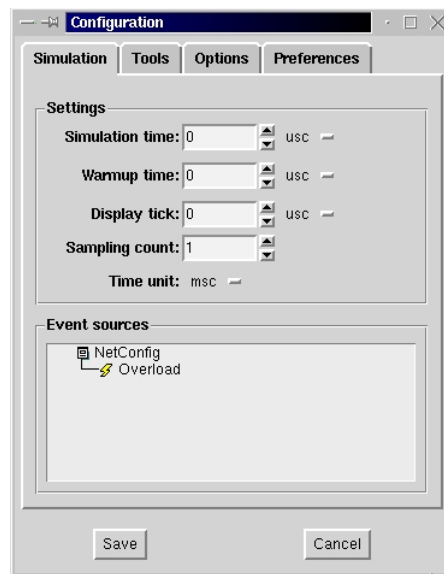
```
# $@timelog
182.867 usc
183.1
184.9
12.5 sec
12.7 sec
```

- ⑩ The **time** setting in **Timer** mode follows the same rules as a simulated time window limit, and specifies a single fixed time for triggering the event.
- ⑩ The **Null** position inhibits the source throughout the simulation.

5.1.4 Generating Events Manually

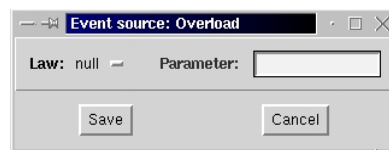
A manual source is linked with a graphic button that can be accessed via the simulation inspector. Each click on this button triggers the service’s routine (C language binding) defined for the associated event. For manual sources, the optional setting is the access path for an output file that will record the list of event start times injected by the user, in a format that can be reused by an automatic source’s file mode.

In our example, the **manual request** event can be configured in manual mode. To do so, use the **Simulation** menu’s **Configure** command. The following window is then displayed:



This window lists the event sources defined in the current simulation configuration under the **Event sources** title, so that the generation laws can be set there. If the **Overload** event does not appear under the **NetConfig** node, you may need to attach this event to the simulation as explained in 3.4.1, Associating Model Instances.

Double-click the icon representing the event source **Overload** so that the following dialog box is displayed:



Choose the **Manual** position without specifying any setting, and then validate by clicking the **Save** button.

5.2 General Simulation Settings

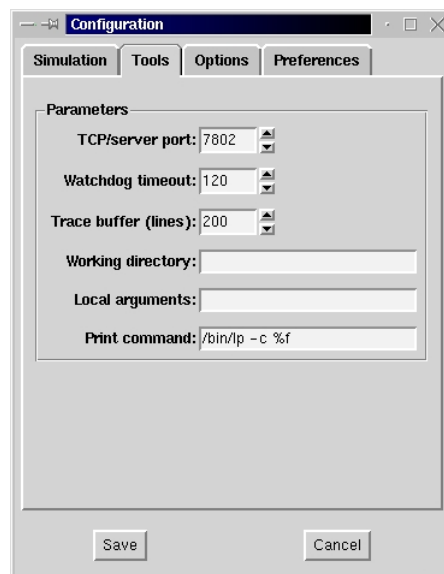
The ISE can be used to set a series of settings relating to the simulation's general behavior. The settings listed under the **Settings** heading of the configuration window discussed above mean the following:

- ⑩ **Simulation time** is the simulation's duration. When the simulated time reaches this value, the monitor permanently stops the simulation.
The value 0 is understood as an indefinite duration. In that case, the user must explicitly stop the simulation.
- ⑩ **Warmup time** defines the startup period during which no statistical samples are taken. This period must correspond to the simulation's transitional initialization period, during which the statistical data is unimportant.
Warning: The total duration of the simulation is made up of the warm-up period added to the **Simulation time** period during which readings are taken.
- ⑩ **Sampling count** is the number of statistical samples that must be taken.
If the value is null, the simulation is considered as having an indefinite duration and the time indicated for **Simulation time** is taken as the sampling period.

- ⑩ **Display tick** defines the smallest time interval that can be displayed.
For example, if the Display tick value is 0.1 μ s, all times will be displayed to the nearest tenth of a microsecond.
- ⑩ **Time unit** is the default unit for displaying time.
For example, if the value of **Time unit** is msc and the value of **Display tick** is 1 μ s, all times will be displayed in the following format: 1,010.005 m. For example, if the value of **Time unit** is sec, the time will be displayed as 1.010005s.

5.3 Simulation Tool Parameters

A second series of parameters can be accessed via the main configuration window **Tools** tab. These can be used to modify certain parameters for the monitor built into the ISE. The window is displayed as follows:

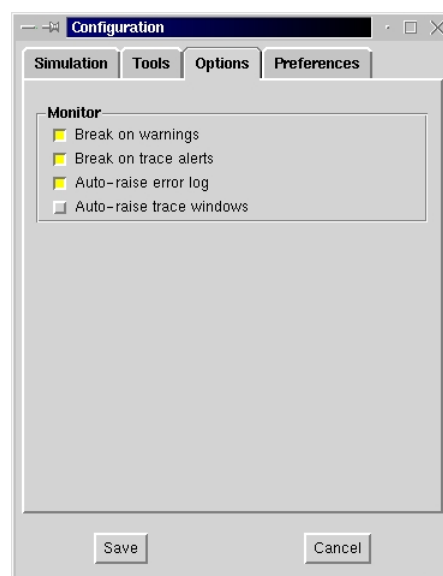


- ⑩ **TCP/Server port** is the TCP/IP port number used for communication between the ISE's monitor and the simulator. The default value was chosen in order to avoid any conflict with the system ports. In addition, as this port only remains active for a very short period of time while a simulation is initialized, it can be shared by several users. Nevertheless, if any conflict occurs this value can be changed using this parameter for the active project.
- ⑩ **Watchdog Timeout** is the time limit during which the connection must be made between the monitor and simulator, in seconds. If this limit is exceeded the ISE simulation startup process is automatically interrupted and an error message is displayed.
- ⑩ **Trace Buffer** is the number of lines of text that can be stored in the monitor's trace results window. A non-null value means that the oldest messages are removed once that number of lines is reached. A null value represents a trace buffer of unlimited size. The default value is set at 200 lines.
- ⑩ **Working Directory** indicates the access path to the simulator's startup directory. This is an optional parameter.

- ⑩ **Local Arguments** is an area specifically for simulator startup arguments that are local to the application. Ideally, these options begin with the reserved prefix (–Q) for this purpose, in order to avoid any conflict with the simulator’s internal options. These options are always added – unmodified – to the end of the list of internal options set by the ISE when the simulator is run. See the “Command line” section for more detailed information on this subject.
- ⑩ **Print Command** contains the command line that will be used to submit print requests to the host system. A system command interpreter is called to run the request. The first occurrence of the “%f” control string is replaced by the name of the file to be printed, if that string appears in the body of the command. In this way, the name of the file that must be printed can be positioned in the command line in the required syntax.

5.4 Operating Options

A final series of options can be accessed via the main configuration window’s **Options** tab. These can be used to enable or disable certain of the simulator’s and ISE’s features. The window is displayed as follows:



The possible settings are:

- ⑩ **Break on Warnings:** selecting this option causes a temporary break in the simulation when the simulator returns a warning. In addition to the temporary break, the message is repeated in the ISE’s **Error log** window, which can be accessed via the **Windows** menu. Such warnings are produced by simulation models calling the SIMEX’s SxManager::warning() method.
- ⑩ **Break on Trace Alerts:** selecting this option causes a temporary break in the simulation when a trace alert is returned by the simulator that has the alert attribute set. Alerts of this type are stored in the API’s trace manager window. Such messages are produced by simulation models calling the SIMEX’s SxTraceManager::writeTrace() method.

- ⑩ **Auto-raise Error Log:** selecting this option causes the **Error log** window containing the simulator's alert messages to be automatically displayed in the foreground as soon as a new message is received.
- ⑩ **Auto-Raise Trace Windows:** selecting this option causes the inspector's objects windows to be automatically displayed in the foreground each time that the ISE holds the simulation.

6. Exporting/Importing the work environment

The exact contents of an FROGS work environment can be exported and then imported again using the **Project** and **Library** menus' **Import/Export** commands. These functions can also be obtained in combination using the ISE's **-e** and **-i** command line options.

Unlike the ISE's command line functions, which produce an export file containing both the project information *and* the models library, the functions accessed via the graphic menus concern either the project *or* the models library.

6.1 Exporting/Importing the current project

Choose **Export** in the **Project** menu. After you have validated the export file's name, every item in the active project is recorded in the export file in portable ASCII format. You can import those items again using the same menu's **Import** function. The imported items permanently replace the active project's existing contents.

6.2 Exporting/Importing the models library

Choose **Export** in the **Library** menu. After you have validated the export file's name, every item in the models library is recorded in the export file in portable ASCII format. You can import those objects again using the same menu's **Import** function. The active models library will be incrementally updated with any missing or modified objects from the import file.

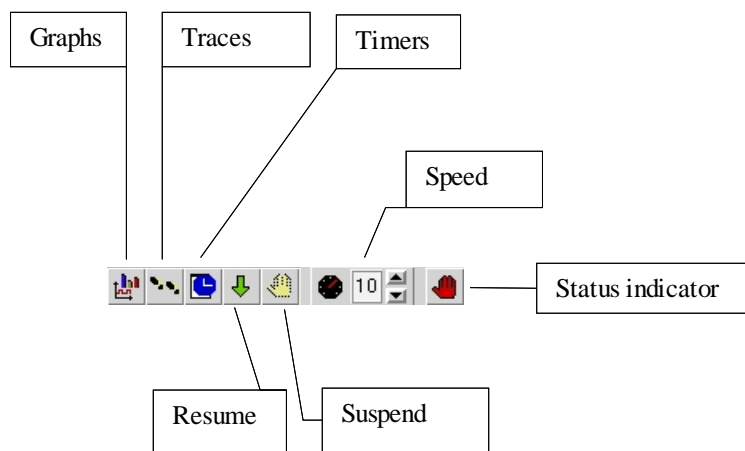
7. Running a Simulation

A FROGS simulation is usually a two-tasks process, running the generated simulator and the FROGS' ISE controlling it. But you can also run the simulator in stand-alone mode, without graphical interface to control it. See 9.2, Simulator Start Options for more on this topic.


7.1 Interactive Execution

The FROGS ISE offers a display and control monitor for handling system objects present in the simulation.

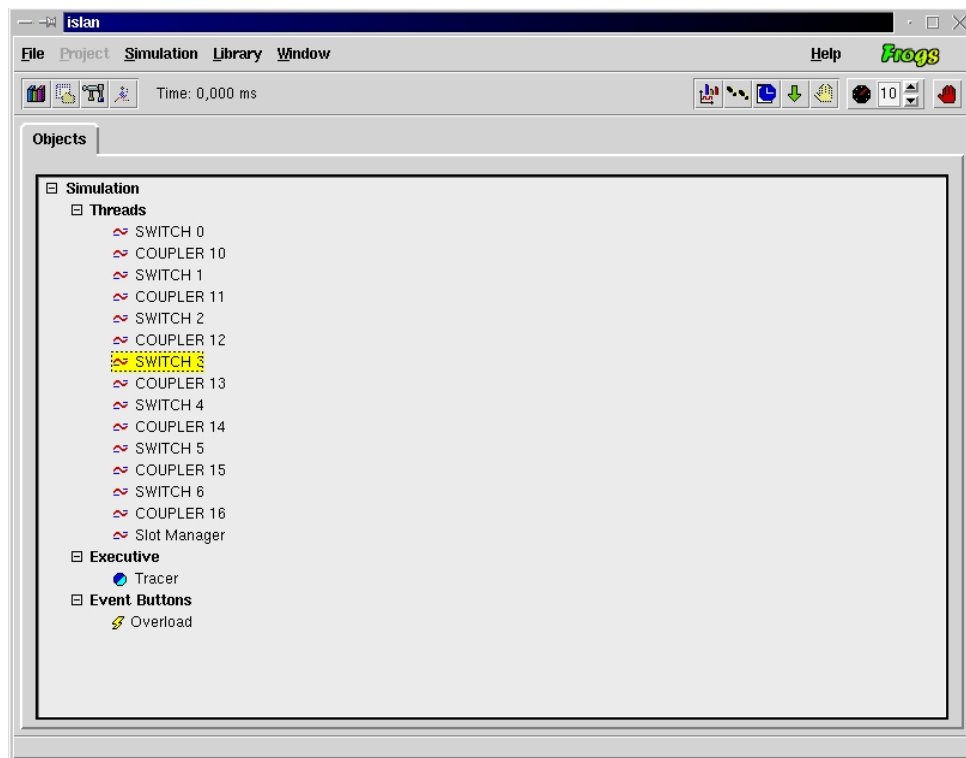
An additional tool displays the functions available from the **Simulation** menu in the main menu bar when simulation is enabled. These functions will be described in detail throughout this section. The tool bar is the following:



7.1.1 Starting Simulation




Starting the simulator is controlled by the **Run** command in the **Simulation** menu. In addition, the  icon in the tool bar is a direct alias for this command. After starting the simulator, the list of system objects is displayed.

In our example "Islan", the following display is generated:










This presentation immediately displays the system objects exported by the simulator.

7.1.2 Controlling Simulation

- ⑩ Simulation is automatically suspended at time value "0" after general simulator initialization. Select the **Release** command from the **Simulation** menu to start execution. This command has an alias in the form of the  icon in the simulation control bar displayed on start-up in the right of the window. You can use this command to resume simulation after any suspension, regardless of the reason (timer, tracing, etc.). After resuming the simulation, the current simulated time shown under the **Time** label in the monitor resumes incrementing.
- ⑩ To suspend simulation again, select the **Hold** command from the **Simulation** menu or click on the  icon in the control bar.
- ⑩ Simulation speed can be controlled using the  selector. Slowing the simulator is achieved using the activity on an internal parasite thread whose sole function is to periodically suspend the simulation process. Reducing the simulation speed does not therefore cause any additional processor resource usage. The maximum simulation speed is obtained when the selector displays a value of 10. At the other extreme, a value of 1 corresponds to the lowest speed.
- ⑩ To definitively stop simulation, select the **Kill** command from the **Simulation** menu.
- ⑩ To restart the simulation, use the **Restart** command from the **Simulation** menu.

7.1.3 Simulator Status Indicator

The current simulator status is shown by an indicator displayed at the extreme right of the monitor tool bar. The pictogram displayed specifies whether or not simulation is active, and in the latter case, why it was suspended. Given the large number of conditions that may warrant simulator suspension, it is useful to refer to this pictogram to determine the type of suspension. The following table summarizes possible status displays:

Pictogram	Status
	Unconditional suspension
	Suspension caused by receiving a warning message
	Suspension caused by a programmed awakening time-out
	Suspension caused by receiving a trace message
	Suspension linked to a breakpoint hit in the graphs
	End of simulation
	Simulation in progress

7.1.4 Using the Inspector

The inspector is an important tool proposed by the simulation monitor. Its function is to present the system objects created and altered as the simulation code is run. It allows interaction with these objects depending on the rules defined by the designer of the simulation models used. The central graphic window in the monitor mode is used by the simulation inspector.

7.1.4.1 Object Organization

FROGS organizes system objects using a hierarchy specified by the simulation model designer.

7.1.4.2 Interaction with an Object

Each object presented has its own graphic representation that is used to display and possibly change its status and/or its characteristics. Double-click on the icon for the **Slot Manager** thread to call-up its interface. If there is none, nothing appears.


7.1.5 Placing Manual Requests

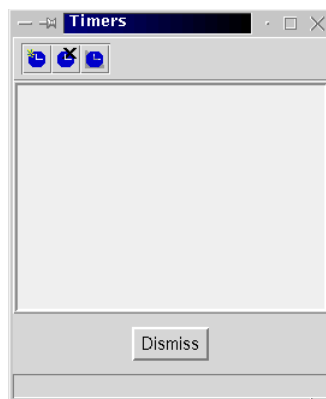
Using the ISE to place manual requests is controlled by a button with a pictogram for each source defined with a manual generation command. These events are simulated as C function calls in the simulator's process. From the simulation inspector's main window, double-click on the **Overload** icon in our example. The following sub-window will appear:




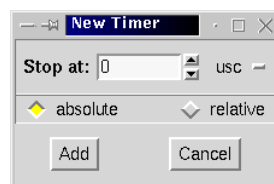
Click on this button to generate an event, i.e. to call-up the service routine assigned by configuration. In our example, the C **startOverloader** function will be activated.



7.1.6 Setting Programmed Awakening Functions



The ISE automatically holds simulation at certain set times. This function is therefore used to stabilize the simulator at a chosen time context prior to analyzing the various system objects. To access the awakening function manager, select the **Timers** command from the **Simulation** menu or choose the  icon from the control bar. The following window is displayed:



To insert a new time for stopping simulation, click on the  icon. A simulated time selection sub-window is then displayed:



- ⑩ Enter the required time, then validate the operation by choosing the **Add** button. The **absolute** and **relative** selectors are used to specify a time-out calculation from time 0 (**absolute**) or from the simulated current time (**relative**).
- ⑩ To inhibit an awakening command without removing it completely, right click on it and select the **Disable** command from the context related menu. For the opposite result, select the **Enable** command from this same menu to reactivate it. You can also left click on it and choose the  icon from the local tool bar.
- ⑩ To toggle the enabled/disabled status of all of the awake commands using a single action, choose the  icon when no other items are selected from the list.

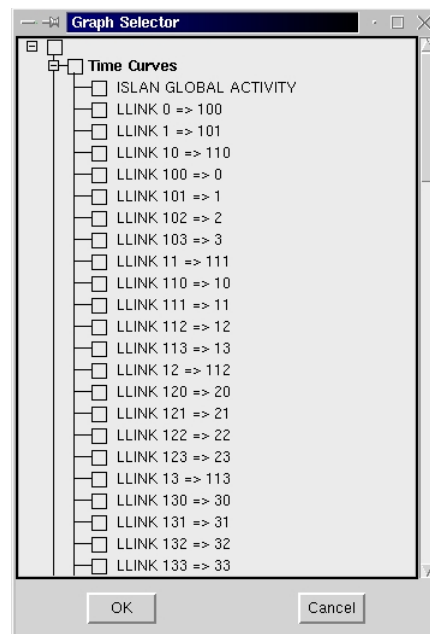
- ⑩ To permanently delete an awake command, right click on it, then select the **Remove** command from the context related menu. You can also left click on it and choose the  icon from the local tool bar.
- ⑩ To delete all awake commands using a single action, choose the  icon when no other items are selected from the list. You will be asked to confirm this command.


8. Displaying Statistics Graphs

Some system objects created by the simulation models export one (or more) displays of their current status to a dedicated plotter, accessible from the ISE monitor. For example, simulation thread transitions are displayed in diagram form, presenting every possible state (suspension, delay, active, etc.) over time. Some models may also propose a graphic illustration of simulated instantaneous processor workload using a curve or a set of histograms showing the distribution of thread states during the simulation. The graphs proposed for display are therefore dependent on the characteristics of the simulation models used. Where necessary, refer to the documentation provided with the execution model used for the meaning of observable parameters.

8.1 Selecting Graphs

To access the plotter functions, select the **Plotter** command from the **Simulation** menu or click on the  icon in the monitor tool bar after starting simulation. The graph selection screen is displayed:



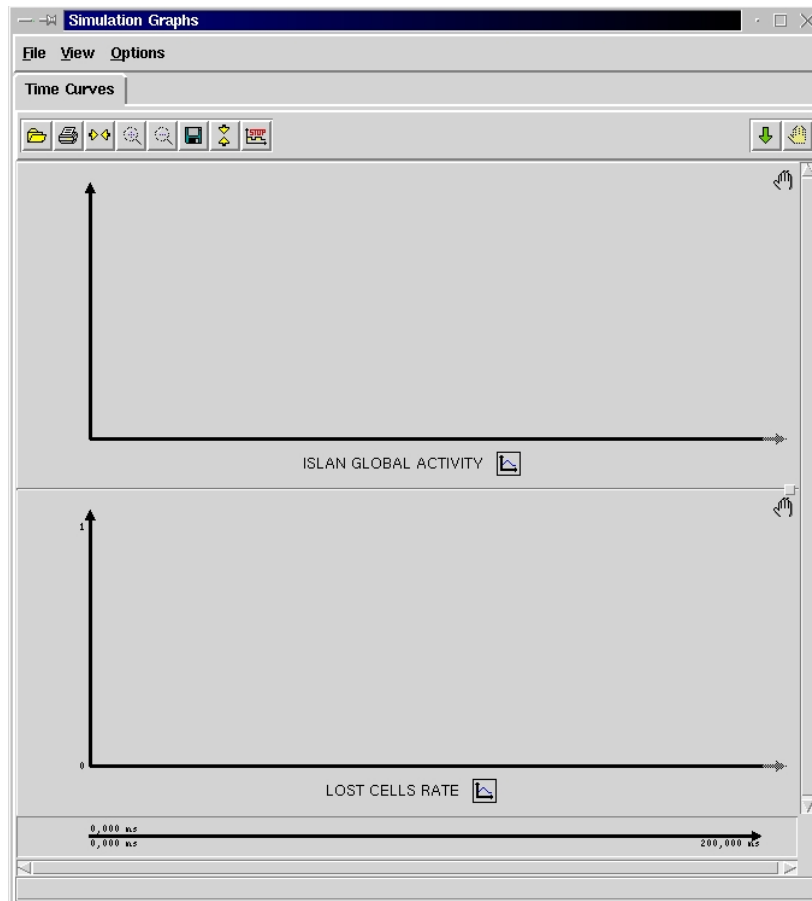
The selection screen only appears when no graphs are currently displayed, otherwise access to the graph trace window is gained directly. At any time you can return to the selector using the **Select** command from the **File** menu in the plotter window or by using the  icon.

As you can see in the previous view, the graphs are listed by graphic type (**Time Curves** for the time curves, **Compounds** for the composite curves and **Histograms** for histograms).

All of the curves that exist at this point in the simulation are available on screen. The selected or not selected state for each node is carried via the lower branches to the graphic objects themselves are reached. The selected curves are displayed in the order of their appearance in the list.

Simulation can export new curves throughout its operation, depending on the performance of the models involved. For example, creating a thread object generally causes the appearance of a new state diagram that represents it. In this case, the graph selector will be enriched on-the-fly.

In our example, choose the **ISLAN GLOBAL ACTIVITY** and **LOST CELLS RATE** curves. After validating this function using **OK**, the plotter window is displayed:



8.2 Display by Type of Graph

8.2.1 Time Curves

Curves indexed to the simulation time are grouped so that they can be compared. The time value is assigned to the abscissa, with a sliding left limit based on its evolution or the current presentation choices. We will use xMin, xCur and xMax respectively as the minimum, current (i.e. simulation time reached prior to the last suspension) and maximum values for this axis in the rest of this document. Click on the **Time Curves** tab in the curve tracer to display this kind of curve.

8.2.2 Composite Curves



A number of time curves of the same type can be assigned together to the same composite curve. They retain their separate properties but appear grouped when displayed in the same graphic field.

8.2.3 Histograms

The histograms are grouped under the **Histograms** tab in the tracer. The Y_axis is used to illustrate the values reached by the different distributions displayed.

8.3 Controlling the Simulation

8.3.1 Continuing/Stopping Simulation

The simulation in progress can be stopped temporarily and restarted from the plotter, using the  and  icons respectively in the control bar.

8.3.2 Setting Breakpoints

Breakpoints can be set on the time curves shown. The transition to one of the states chosen from a State Diagram, or reaching a numerical limit in a simple state diagram can therefore trigger an automatic stop to simulation. Control is then returned to ISE, which then synchronizes all of the views available on the breakpoint such as updating the status of inspector objects.

To access this function, right click on the title of a time curve to call up the local control menu for this curve, pulldown the **Mode** sub-menu, then choose **Breakpoint**.

8.3.2.1 Breakpoints in a State Diagram


Once the **Breakpoint** function is validated, a list of possible diagram states is displayed in a selection sub-window. Check the selector for each status required. The **Close** button ends the selection process.

8.3.2.2 Breakpoints in a Simple State Diagram

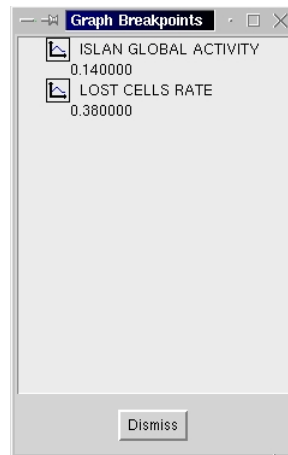
Once the **Breakpoint** function is validated, a sub-window is displayed for entering the threshold value. Enter this value, which will stop the simulation when it is reached. The **Close** button creates the breakpoint for the current threshold value. The **Remove** button deletes the breakpoint for the displayed value.

An accelerated mode for setting breakpoints can be accessed by directly double-clicking on the chosen part of the curve, representing the ordinate value for the status or threshold.

8.3.3 Displaying and Checking Breakpoints

A function lets the user gain a summary overview of all of the breakpoints set on the time curves. This function will also allow disabling or removing these breakpoints, either individually or curve by curve. Use the **Breakpoints** option from the **View** menu or click on the  icon to access this function.

A window like the one below is displayed:





The breakpoints displayed are grouped by the curve they belong to. Right click on a breakpoint value to pulldown the local control sub-menu. The **Remove**, **Disable**, and **Enable** choices respectively will remove, disable or re-enable the selected breakpoints. If you would like to apply these actions globally to all of the breakpoints on a given curve, right click on the name of the curve as displayed in this same window and use the required sub-function.

The **Close** button closes the breakpoints control sub-window.


8.4 Scale Compression


8.4.1 Y Compression

When the initial height of the curves does not allow their complete display and requires the use of the vertical scroll bars, then it is sometimes useful to vertically compress these objects. Use the **Y-compress** function in the **View** menu or the  icon to reduce the height of these objects so that they can be displayed on a single graphic page. Note that this option is automatically disabled if too many objects are already compressed for display. This option is available for time curves and histograms.


Vertically uncompressing the display, i.e. returning to the previous display scale is possible using the **Y-uncompress** function from the **View** menu or by using the  icon.

8.4.2 X Compression


It is often convenient to be able to display what a curve looks like over the entire simulation time range. Use the **X-compress** function from the **View** menu or the  icon to bring all of the stored points from all of the time curves back to a single graphic page. This operation is performed by applying an abscissa scaling function that changes the displayed time/pixel ratio. The left and right limits of the curve are brought back to 0 and xCur respectively.

Horizontally uncompressing the display, i.e. returning to the previous display scale is possible using the **X-uncompress** function from the **View** menu or by using the  icon.

8.4.3 Zoom In

The zoom in function expands the time/pixel scale by 200%. The left hand limit (xMin) is retained as closely as possible, while the display is recalculated for all of the time curves displayed. This function is a gradation of the X compression function described previously. It is called up by choosing **Zoom In 200%** from the **View** menu or using the  icon.


8.4.4 Zoom Out

The zoom out function contracts the time/pixel scale by 50%. The right hand limit (xMax) is changed to match the new scale, while the display is recalculated for all of the time curves displayed. This function is symmetrical to the zoom in function described previously, but is however applied using a lower scaling factor. It is called up by choosing **Zoom Out 50%** from the **View** menu or using the  icon.

8.5 Composite Curves

In order to obtain a single curve made up from the states of a number of other compatible curves (i.e. curves of the same type and in the case of state, with the same states), it is possible to use the "drag & drop" approach based on the time curves title bar. To do this, simply left click on the title bar of one of the curves to group, then drag the mouse to the title bar of the other curve to group, whether it is a composite curve or not, then release the left mouse button. In the former case, the first curve will be added to the second composite, in the latter case, the two curves will form a new composite. In this latter case only, the tracer will prompt the user for the name of the newly created composite curve.

8.6 Placing Graphs

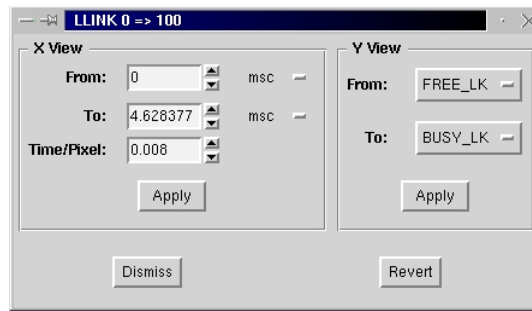
Graphs may be arranged for display using a simple placing mechanism, triggered by a "drag & drop" operation performed on the  grab icon provided on each one. The graph located at the source of the move action will be inserted before the one located at the move destination.

8.7 Selection and Cross Hairs

8.7.1 Actions on Graph Sections

Some graph analysis actions require expanding or compressing a specific part of the time curve trace. The graphic selection concept is used to select a precise area of application for a change of scale. To call up this function, right click on the title of a time curve to call up the local control menu for this curve, pulldown the **Mode** sub-menu, then choose **Selection**.

A sub-window like the one shown below will then be displayed:



The **X-View** heading in this window lets the user control the time/pixel ratio (i.e. the abscissa) over the current selection. The **Y-View** heading is used to control the field shown by the ordinates.

In parallel with the window display, a selection rectangle is shown on the selected curve. Bring this rectangle over the graphic portion that you wish to assign to the selection. To do this, left click on the point that represents the top left corner of the rectangle, then drag the pointer to the lower right corner, while holding the mouse button down. The selection made is determined by the rectangle formed when the left mouse button is released. You can rework an existing selection by holding down the **CONTROL** key while performing the operation using the left mouse button.

During this operation, the current limits of your selection are displayed in real-time in the control window, showing both the abscissa and ordinate values. You can change these limits manually using the data entry fields provided.

In the **X-View** heading, **From** and **To** respectively control the xMin and xMax values displayed. **Time/Pixel** is the display ratio between a graphic pixel and its time correspondence.

In the **Y-View** heading, the system will display the limits-states for a State Diagram, or the minimum and maximum values in ordinates, for a simple state diagram. In the former case, only those states present between the two limits (inclusive) will be displayed. In the second case, only those points between the minimum value and the maximum value (inclusive) will be displayed.

Apply the new display parameters by clicking on the **Apply** button. To revert to the previous state, use the **Revert** button.

There is a quick way to access the selection function, by simultaneously pressing the left mouse button and the **SHIFT** key while directly selecting the target curve. Once the selection has been made, right click on the chosen curve, then validate **X-Apply** or **Y-Apply** respectively to change the scale or the corresponding limits.

8.7.2 Using the Cross Hairs

Left click on a point on a time curve or a histogram to see its coordinates. The coordinates will be displayed at the top of the selected curve. The cross hairs let you move along this curve while holding down the left mouse button, for a real-time display of the corresponding coordinates.

8.8 Other Local Functions

The graphs shown all have a local menu that lets the user access a set of individual functions. To access this menu, right click on the title of the graph.

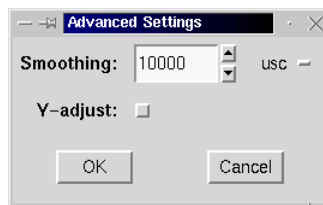
8.8.1 Local Time Curve Functions

8.8.1.1 Seeking the Next Point

The **Seek** sub-function in a curve's local menu lets the user seek the next or the previous point received from the simulation by looking forward (**Forward** input) or backward (**Backward** input) within the current view. This function is useful when looking for the next or the previous transition in a State Diagram. The minimum limit xMin is moved to bring the point found onto the display. If no point is found, an audible beep will sound, indicating that the seek action has failed to find anything.

8.8.1.2 Advanced Settings

Simple state diagrams (i.e. time curves excluding state diagrams) have an additional setting feature that can be accessed from the **Advanced** sub-function in their local menu. The advanced settings sub-window looks like this:



The **Smoothing** parameter is a smoothing constant applied to the graph along its horizontal scale. It is expressed as a simulated time value. Selecting the **Y-adjust** parameter will automatically adjust the maximum ordinates limit according to the points received from the simulator. If not, the limit will not be readjusted to match the maximum value received from the simulator.

Choose **OK** to validate the changes made to the settings, or **Cancel** to cancel the operation.

8.8.2 Local Histogram Functions

8.8.2.1 Display Mode

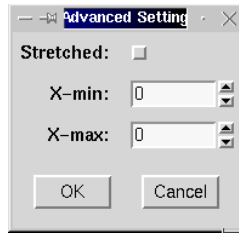
The histogram display mode can be controlled by the **Display** sub-function. The **Density** option specifies the probability density, the **Repartition** option specifies the repartition function. By default, the probability density is applied.

8.8.2.2 Representation Type

The histogram representation type can be controlled by the **View** sub-function. The **Absolute** option specifies the absolute mode, the **Relative** option specifies the relative mode. By default, the relative mode is applied.

8.8.2.3 Advanced Settings


All histograms have a additional settings that are accessed by the **Advanced** sub-function from the local menu. The advanced settings sub-window looks like this:



Selecting the **Stretched** parameter will extend the histogram trace surface to cover all of the drawing surface available along the horizontal axis. Otherwise, a default horizontal size is given to the graph, which may vary depending on the distribution presented.

"X-min" and "X-max" respectively are the simulated time minimum and maximum values used to represent the distribution.

8.8.2.4 Updating

Unlike time curves, histograms are not updated in real-time during simulation. Updating can be triggered on demand using the **Update** function in the **View** menu or using the  icon.

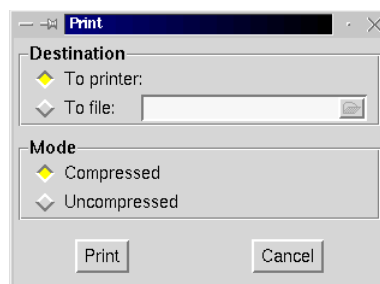
8.8.3 Common Functions

8.8.3.1 Changing Colors

The color used to draw the graphs may be changed using the selector that is accessed via the **Color** sub-function in the local menu for these graphs.


8.8.3.2 PostScript Printing

Time curves and histograms can be printed out in PostScript format, either to a file or directly to the chosen printing peripheral. From the graph's local menu, use the **Print** sub-function to access the Print sub-window. The following window is displayed:




Choose the destination, **To printer** for a direct print out to a printer, **To file** to route the print out to a file. In the latter case, the filename field next to the selector must be filled-in. For a direct print out, the current **Print command** settings in the simulation configuration parameters are used. These settings are accessible from the **Tools** page in the **Configure** window in the ISE **Simulation** menu.

Horizontal curve compression is applied by default, so as to present the entire simulated time range in the PostScript trace. You may disable this option by choosing the **Uncompressed** mode.

A global print command for all of the graphs present on the current page (time curves or histograms) can be called up from the **Print** command in the **File** menu or using the  icon.

8.8.3.3 Removing the Graph

The **Remove** sub-function, accessed from the local menu for a graph lets the user remove it from the display, regardless of its type. It may be restored using the graph selector accessed via the **Select** command in the **File** menu or via the  icon.

8.9 General Options

The plotter offers a number of options that affect its overall performance. These options are accessible via the **Options** menu.

8.9.1 Adjusting Abscissas

The **X-adjust** mode is used to choose automatic readjustment of the xMin and xMax limits when running the time curves, depending on the minimum and maximum values received from the simulator. This allows retaining only one active graphic page at any time during the simulation, and therefore avoids the need to use the vertical scroll bar. This mode is disabled by default.


8.9.2 Scroll Lock

The **Scroll lock** mode indexes time curve horizontal scrolling to mouse motion, when using the horizontal scroll bar. Otherwise, the display is only refreshed when the mouse button is released. This mode, when it is enabled, requires a high trace speed from the host station, due to the numerous display refreshes that take place. This mode is disabled by default.

8.9.3 Auto-Select Color

The **Auto-select color** mode lets the tracer choose the colors to assign to new curves. This mode is initialized by default.

8.9.4 Auto-Save Session

All of the current tracer session parameters can be saved on demand using the **Save** command from the **File** menu or using the  icon. This includes the list of curves displayed, the composite curves that are active, the position of the breakpoints on the different curves, the advanced configuration settings, etc. The **Auto-save session** option lets the user choose to automatically run this command at the end of each session, when the simulator is shutdown.

9. Command Lines

9.1 ISE Start Options

The FROGS ISE has the following set of start options available:

- ⑩ **-f <project-file>**, automatically loads the simulation project designated by its access path when the ISE is initialized. This option will override the default behavior to pre-load the last active project.
- ⑩ **-R <repository>**, gives the access path to the ISE default models database. This value is only used when the ISE is first started in a user environment to build the initial database. This option will override the default value set to **<private-dir>/frogs.fdb**, where **<private-dir>** is the name of the directory created by the ISE during the very first user session. It is located in the root of the user's account and is called **.frogs/<arch>** where **arch** is the identifier for the current system architecture.
- ⑩ **-x**, forces simulation to start, as soon as ISE initialization is finished. Combined with the **-f** option, this selection allows the automatic execution of any simulation project. The **-x** option is mutually exclusive with option **-p**.
- ⑩ **-p**, is an internal option used by the simulator to run a **slave** monitor interface when simulation is started using a command line.
- ⑩ **-q**, unconditionally shuts down the ISE as soon as simulator execution is finished. By default, the ISE remains active after the end of a simulation, except in the case of the **slave** mode designated by the **-p** option.
- ⑩ **-u**, inhibits reloading of the last ISE working context, especially its last current project. By default, the ISE restores the last project, along with some general display characteristics like the geometry of the graphic window that it uses.
- ⑩ **-v**, sends the ISE version number to the standard output.
- ⑩ **-e <export-file>**, used in combination with the **-f** option allows exporting the content of the specified project in ASCII format along with the model manager database attached to this project. In other words, the ASCII file obtained will contain all of the information required to rebuild an identical FROGS working environment. This rebuild action can be performed using the **-i** option in the ISE command line or by using the **Library** menu input function provided in the same tool. The ISE will automatically end its execution after exporting.
- ⑩ **-i <import-file>**, used in combination with the **-f** option allows importing the content of the specified project in ASCII format along with the model manager database attached to this project. This operation is symmetrical to the export function obtained using the **-e** option. If the project specified using the **-f** option does not exist, it is automatically created. If not, its content is replaced by the components found in the imported file. The model manager database is incrementally updated with the missing or modified elements found in the imported file. The ISE will automatically end its execution after importing.

9.2 Simulator Start Options

Any simulator generated using the standard procedure described in chapter 3 includes the following set of start options:

- ⑩ **-C <config>**, specifies the name of the simulation configuration to run. This option is ignored if **-f** (or **-F**) is used. In the latter case, the active configuration name is taken from the project file stated as the argument.
- ⑩ **-R <repository>**, specifies the access path to the model database from which the simulation configuration designated by the **-C** option will be taken. This option is ignored if **-f** (or **-F**) is used. In the latter case, the active configuration name is taken from the project file stated as the argument. If no option sets the access path to the model database, the **<private_dir>/frogs.fdb** file will be used, where **<private_dir>** is the directory name created by the ISE when the very first user session is run. It is located in the root of the user's account and is called **.frogs/<arch>** where **arch** is the identifier for the current system architecture.
- ⑩ **-F <projfile>**, initializes the simulator using configuration parameters found in the project file stated as the argument, but without attaching the ISE to the simulation. This mode enables the matching of the model database and configuration files to be used, ready for running without a graphic interface.
- ⑩ **-f <projfile>**, triggers running the ISE in "slave" mode, allowing graphic interaction with the simulator. The argument entered is the access path to the project file that must be pre-loaded by the ISE. This file will also provide the information on the model database and the configuration to be used.
- ⑩ **-X** is a sub-option introducer used for setting the simulator's Boolean parameters. This option must be followed by the list of flags to be activated. Valid flags include:
 - ⑩ **« w »**, causes automatic simulation suspension on receiving alert messages. This option is only active during interactive simulation, in the presence of the graphic monitor. By default, message type reception does not suspend simulation.
 - ⑩ **« a »**, causes automatic simulation suspension on receiving a trace with the alert attribute set. This option is only active during interactive simulation, in the presence of the graphic monitor. By default, message type reception does not suspend simulation.
 - ⑩ **« m »** activates a conservative, slower, management mode for the multi-threaded executive that is internal to FROGS. Running this mode is generally imposed for any use of the simulator in parallel with a memory analysis application that works by instrumenting the application's binary.
- ⑩ **-t <time[s,m,u]>**, specifies the simulation time limit (simulated time). By default, the simulation duration is infinite. This option is equivalent to the ISE **Simulation time** parameter and applies to the project.
- ⑩ **-w <time[s,m,u]>**, specifies the simulation warm up period before any statistical measurements are made. This option is equivalent to the ISE **Warmup time** parameter and applies to the project.
- ⑩ **-s <nsamples>**, states the number of statistical samples that will be collected by the simulator during the simulation duration. If this parameter is null, then a single sample will be collected at the end of the simulation. If the simulation duration is infinite, no sampling will be performed. In all cases, the statistical samples will be taken periodically at a frequency that equals the simulation duration divided by the number of samples required. By default, only one sample will be taken.

- ④ **-d <directory>**, places the simulator in the directory specified in the parameter prior to execution. By default, the current directory is retained.
- ④ **-l <errlogfile>**, specifies the access path to the file used to store the warning messages generated by the simulator. By default, the messages are sent on the simulator's standard error output.
- ④ **-v**, used along in the command line, this option sends the version number of the FROGS simulator on the standard output. When used with start options, this command will cause the display of type and version information for the different simulation models that are instantiated.
- ④ **-z <speed>** , controls the simulation speed using a factor that runs from 1 to 10. 1 corresponds to the maximum slowing factor.