TOPO: Quick Reference
Front-End — version 3R6

José A. Mañas
Tomás de Miguel
Tomás Robles
Joaquín Salvachua
Gabriel Huecas
Marcelino Veiga

Dpt. Ingeniería Telemática
E.T.S.I. de Telecomunicación
Univ. Politécnica de Madrid
E-28040 Madrid, SPAIN

<topo@dit.upm.es>

18 January, 1995

Abstract

TOPO is a set of interoperating LOTOS tools grouped in several packages. This one is basically the front-end, including the basic facilities (mainly syntax and semantic checking).

This paper is a quick reference guide for users.

1 Getting Started . . .

TOPO is a set of interoperating LOTOS tools grouped in several packages. This one includes mainly syntax and semantic checking, and it is a front-end for the other ones.

spec`.lot` → The LOTOS specification

There are basically two shell tools to drive the concrete components:
`toposet`: sets a compilation environment.
`topo`: coordinates compilation activities.

2 Environment

TOPO tools are accessed via a unique interface called `topo`, that is described in the next section. There are some extra files that are needed at compile time. The paths are coded relative to the value of a root directory that is fixed at installation time. However, the paths may be affected by the settings of the following environment variables:

variable	default	used for
TOPO	user	root for topo tools
TOPOBIN	$TOPO/bin	binaries to be executed
TOPOINC	$TOPO/lib	files to be included
TOPOLIB	$TOPO/lib	libraries to be loaded
TOPOSTDLIB	$TOPO/stdlib	lotos libraries to be loaded

Variable TOPO is mandatory.

The other variables may be set only if needed. If these variables are not found in the environment, `topo` sets sensible values for them and, only in the case of UNIX, puts them in the environment for called tools to benefit from. UNIX version sets as well the PATH.

In MS-DOS, the user is expected to set environment as desired. `topo` will just read it for internal purposes.

You may further influence the behaviour of concrete tools by setting environment variables. There is one variable for most tools in the package. See the corresponding QuickReference document.

These are the relevant ones for the frontend

variable	tool	help	example
TOPOF_LFE	lfe	lfe -h	setenv TOPOF_LFE -c
TOPOF_LSA	lsa	lsa -h	setenv TOPOF_LSA -f

3 Context

A number of compilation options may be specified in a context that is stored a a separate file (spec.ctx). It is set by `toposet` and used by `topo`. It may be edited, but be very careful to strictly respect the format, since there is no error checking at all!

`toposet` is expected to be used before `topo`. It stores the current context into SPEC.ctx. If there is no such file, `topo` will use some default values. `toposet`, if executed without options, sets a default context:

```
Library
Language C
BehaviourName lbc
DataName ldc
BehaviourPieces 1
DataPieces 1
GladLibrary
Makefile
```

`toposet` spec[`.lot`] [options]	
-help	One line help
-context F	Use file F.ctx as context
-make F	use F as makefile
-list	Print current context
-verbose	Echo activity
-library L	IS8807 defines only one library of predefined data types, *THE Standard Library*. Nevertheless, TOPO permits to use a different library. Users may use their own libraries (any LOTOS specification).
-nolibrary	Don't use any standard library
-tname N	Root name for generated files (behaviour)
-tpieces N	Number of pieces to break C files down
-dname N	Root name for generated files (data)
-dpieces N	Number of pieces to break C files down
-glad G	GLAD is a tool to annotate data types, so that sensible code may be generated from them. Its role may range from selecting the naming of the LOTOS data objects when code is generated, to linking to external data types. The most basic use of GLAD allows to choose internal names for the operations when code is generated (cryptic but solid) `toposet spec -glad $TOPOSTDLIB/internal.gld` or to use the lexical values (nicer, but risky): `toposet spec -glad $TOPOSTDLIB/lexical.gld` Users may specify their own annotation criteria.
-noglad	do not use GLAD.
-language L	Either C or Ada

The default is to use no standard library. The following *standard* libraries are available:

minimal	→ Only *Booleans* (default)
ditupm	→ improved and extended IS library
is	→ The standard library as in IS8807
is-neq	→ idem, but with no equation
mod-is	→ IS8807, modified by LotoSphere
mod-is-neq	→ idem, but with no equation
bool_nat	→ Only *Booleans* and *Naturals*
<>	→ User defined at `./<>.lot`

`toposet` plays an active role in removing intermediate compilation results that are no longer valid or needed upon option changing. It revises as well the coherency of the standard library (if any), and issues a warning if it is out of date and tries to regenerate it.

More precisely, `toposet` tries to locate the desired library in the current directory. If it founds "`./LIB.lot`", it checks if file "`./LIB.ls`" exits and is newer, issuing a warning if fails and trying to regenerate it. Otherwise, if there is a "`./LIB.ls`", this is recorded as standard library to use. Lastly, if the previous checks fail, `toposet` tries "`TOPOSTDLIB/LIB.ls`"; if it is found, it is recorded as the standard library to use. If all the previous ones fail, `toposet` emits an error.

Later on, `topo` will use the recorded library. If there would be a "`./LIB.lot`", it will check consistency in every compilation, to check for updates of the "standard" library.

4 Actions

`topo` is a collection of interoperating tools. All of them may be accessed via a single interface that coordinates them for effective interworking. Command line flags are used to select the intended functionality. This paragraph takes care of basic actions.

`topo` spec[`.lot`] [action]	
-help	On line help
-syntax	Just checks syntax conformance.
-s	Checks semantics conformance and builds user libraries of data types.
-listmake	shows internal makefile to control tool interoperation
-clean	Clean directory: intermediate files are removed.
-tidy	More drastic cleaning: all the generated files are removed.
default	-s

It is also possible to provide options to `topo`.

`topo` spec[`.lot`] [topo_options] [action]	
-context F	Use file F.ctx as context
-verbose	Echo activity

And even to actions.

`topo` spec[`.lot`] [topo_options] [action [action_options]]