

SPELL Build Manual

SPELL version 2.4.4

Distribution list

Full Report:

SES-SSO-SOE

For Information Purposes:

Open Source

Reference:

SES-SSO-SOE-SPELL-2015/01

Date issued:

February 2015

Acronyms

CV	Command Verification
GCS	Ground Control System
GDB	Ground Database
GUI	Graphical User Interface
HMI	Human Machine Interface (equivalent to GUI)
IDE	Integrated Development Environment
MMD	Manoeuvre Message Database
OOL	Out-of-limits
PDF	Portable Document Format
PROC	Automated SPELL procedure
RCP	Rich Client Platform
S/C	Spacecraft
SCDB	Spacecraft Database
SDE	SPELL Development Environment
SEE	SPELL Execution Environment
SES	Société Européenne des Satellites
SPELL	Satellite Procedure Execution Language and Library
TC	Telecommand
TM	Telemetry
URI	Uniform Resource Identifier
USL	Unified Scripting Language
UTC	Coordinated Universal Time

Table of Contents

1	Introduction	4
1.1	Purpose of this document.....	4
1.2	Software Requirements	4
1.3	The Build System	5
2	Building SPELL	6
2.1	Directory Structure	6
2.2	Preparation	6
2.3	Configuration	7
2.4	Compilation	10
2.5	Installation	10
2.6	Cleanup.....	11
3	Building SPELL DEV.....	12
3.1	Directory Structure	12
3.2	Preparation	12
3.3	Configuration	13
3.4	Compilation	15
3.5	Installation	15
3.6	Cleanup.....	16

1 Introduction

1.1 Purpose of this document

This document is the SPELL build user manual for the SPELL framework. It is intended to be used by software teams in order to build and deploy the framework components in a GNU/Linux platform.

The framework is divided in two separate projects, SPELL (including the language library, the execution environment and the GUI client) and SPELL DEV (the development environment).


1.2 Software Requirements

The following packages and libraries are needed to build the SPELL framework:

Package	Version	Remarks
GNU autotools	1.9	
GNU GCC buildchain	4.1.2	
GNU Bash	3.1	
GNU Binutils	2.16	
GNU Make	3.8	
POSIX threads (libpthread)	2.4	
POSIX 1.b realtime extensions (librt)	2.4	
BSD utility functions (libutil)	2.4	
Xerces-C (libxerces-c)	27	
Dynamic linking interface (libdl)	2.4	
C/C++ standard libraries (libstd)	6.0	
Python library and headers (libpython)	2.5,2.6	Not compatible with 2.7+
NCurses development libraries (libncurses) (**)	5.5	
Log4CPlus	1.2	log4cplus.sourceforge.net
Eclipse IDE	3.7	Not compatible with 3.8+
Java JDK	1.6	Not checked against 1.7+
Subversion(*)	<1.7.0	

(*) The subversion package is required only for the SPELL database support for subversion-controlled files. It is not mandatory to execute procedures.

IMPORTANT: in order to correctly build the Eclipse RCP based applications, it is required to have a standard Eclipse installation prepared for *Plug-in Development*. This means that all plugins related to the PDE (Plug-in Development Environment) shall be available in the Eclipse platform. Please check www.eclipse.org for details.

03 February, 2015	SPELL Build Manual	
Page 5 of 16	File: SPELL - Build Manual - 2.4.4.docx	

1.3 The Build System

The GNU auto-tools have been chosen as the build system for SPELL, since they provide a widely-used, GNU-standard way of compiling, configuring and deploying software in GNU/Linux platforms. Please refer to <http://www.gnu.org/software/autoconf/> and <http://www.gnu.org/software/automake/> for details.

In the following sections, a basic knowledge of what auto-tools are and how do they work will be assumed.

2 Building SPELL

In this chapter, the steps for building and installing the SPELL execution environment are described. It is assumed that the source code has been checked out from our web site repositories.

In the following, it is assumed that the source code is checked out in a directory named "workspace".

2.1 Directory Structure

Below, the basic list of directories and files in the `workspace` directory can be found:

<code>common.mk</code>	<code>(autotools)</code>	Common configurations for Makefiles
<code>config</code>	<code>(spell)</code>	SPELL components configuration files
<code>configure.ac</code>	<code>(autotools)</code>	Main autotools configuration script
<code>data</code>	<code>(spell)</code>	SPELL components input files
<code>doc</code>	<code>(spell)</code>	SPELL documentation
<code>drivers</code>	<code>(spell)</code>	SPELL driver modules
<code>include</code>	<code>(spell)</code>	SPELL software header files
<code>lib</code>	<code>(spell)</code>	SPELL libraries source files
<code>m4</code>	<code>(autotools)</code>	M4 macros for autotools
<code>Makefile.am</code>	<code>(autotools)</code>	Makefile description input for autotools
<code>py-compile</code>	<code>(spell)</code>	Script for compiling Python files
<code>py-compiledir</code>	<code>(spell)</code>	Script for compiling Python modules
<code>rcp</code>	<code>(spell)</code>	Eclipse RCP 3.5 libraries
<code>scripts</code>	<code>(spell)</code>	Scripts for launching some SPELL programs
<code>src</code>	<code>(spell)</code>	SPELL programs source files

This set of files and directories corresponds to a completely clean SPELL source code package, as it comes in the source distribution. This package is still not prepared for build, since the auto-tools working files have not been generated yet. They are generated in the *preparation stage*, described in the next section.

2.2 Preparation

First the auto-tools working files have to be generated. This is done by running the "autoreconf" command in the base directory of the SPELL source code:

```
$> cd workspace
$> autoreconf -fi
```

This program will parse the *configure.ac* main script, and will create and install the standard files required for the compilation. The list of files and directories after running *autoreconf* should be something like this (the new files and directories are highlighted):

aclocal.m4	(autotools)	Autoconf M4 macros
autom4te.cache	(autotools)	Cached data used by autoconf
common.mk	(autotools)	Common configurations for Makefiles
config	(spell)	SPELL components configuration files
config.guess	(autotools)	Utility script for autoconf
config.h.in	(autotools)	Input file for the standard config.h
config.sub	(autotools)	Utility script for autoconf
configure	(autotools)	Main build configuration script
configure.ac	(autotools)	Main autotools configuration script
data	(spell)	SPELL components input files
depcomp	(autotools)	Utility script for autoconf
doc	(spell)	SPELL documentation
drivers	(spell)	SPELL driver modules
include	(spell)	SPELL software header files
install-sh	(autotools)	Utility script for autoconf
lib	(spell)	SPELL libraries source files
ltmain.sh	(autotools)	Utility script for autoconf
m4	(autotools)	M4 macros for autotools
Makefile.am	(autotools)	Makefile description input for autotools
Makefile.in	(autotools)	Intermediate makefile data
missing	(autotools)	Utility script for autoconf
py-compile	(spell)	Script for compiling Python files
py-compiledir	(spell)	Script for compiling Python modules
rcp	(spell)	Eclipse RCP 3.5 libraries
scripts	(spell)	Scripts for launching some SPELL programs
src	(spell)	SPELL programs source files

Notice that additional "Makefile.in" files will be created everywhere in the directories where a "Makefile.am" file can be found. At this point, the source code is prepared to be configured with specific build options, and then compiled.

2.3 Configuration

Before compiling the source code, the build has to be configured in order to specify some options. These include the installation target directory, the location of some library dependencies, and the enablement flags for the SPELL components.

To configure the build with specific options, the "configure" script generated by *autoreconf* is to be used. This script follows the standards for building and compiling any GNU software.

The "configure" script can show a description of all available options by running it with the "--help" argument:

```
$> ./configure --help
```

Part of the command output is presented below:

```
`configure' configures SPELL to adapt to many kinds of systems.
Usage: ./configure [OPTION]... [VAR=VALUE]...

...
--enable-executor          Compile the executor process (default yes)
--enable-executorcmd       Compile the command line executor process (default no)
--enable-gui               Compile the SPELL-GUI (default no)
--enable-driverstd         Compile the SPELL Standalone driver (default yes)
--enable-shell             Compile the SPELL shell (default yes)
--enable-documentation     Compile the documentation (default no)
--enable-listener          Compile the listener process (default yes)
--enable-context           Compile the context process (default yes)
--enable-procs             Deploy sample procedures (default no)

...
--with-eclipse=PATH       Provide eclipse installation used to compile RCP components
--with-log4cplus=PATH     Provide the path to the logger libraries
...
```

Only the parameters specific to SPELL (not GNU standard) have been shown.

All the "--enable" parameters can be used to enable or disable the compilation (and installation) of concrete SPELL components like the SPELL-GUI, or the SPELL drivers, for example. These parameters accept the values "yes" or "no" to respectively enable or disable the associated component.

The table below shows the available components and their default enablement state:

Component	Enabled by default
Executor	YES
Listener	YES
Context	YES
Command line executor	NO
GUI	NO
Standalone driver	YES
Shell	YES
Sample procedures	YES
Documentation	NO

There is also one mandatory "`--with-eclipse`" argument. The user shall provide the path to an existing Eclipse® IDE installation with this argument, otherwise the build process will fail. Another configure parameter, this time part of the GNU standards, is "`--prefix`". This parameter can be used to specify the final location where the SPELL framework will be deployed after compilation. If this parameter is not given, the SPELL framework will be installed following the GNU standards.

IMPORTANT: as of the current revision, it is required to use the "`--prefix`" parameter to specify an installation directory for SPELL. This way, all the SPELL components will be installed together under the same location. If this parameter is not given, the software will be distributed in different directories of the operating system (`/usr/bin`, `/usr/lib`, etc.) and will break the SPELL execution environment setup.

WARNING: you may need to specify the path to the Python shared library file. If that is the case, the configure script will warn you about it and tell you how to solve the problem.

Please refer to the "`configure`" script help for details about this parameter and others.

An example of a complete command to configure the build is shown below:

```
$> ./configure
      --prefix=/home/spell/SPELL
      --enable-gui=yes
      --with-eclipse=/opt/eclipse-3.7
```

This command will build the components Executor, GUI, drivers, Shell and will use the Eclipse® installation located in "`/opt/eclipse-3.7`" directory.

As a result of the "`configure`" command, a set of "`Makefile`" files are generated. These make files are the final inputs for the GNU make program, that will perform the software compilation.

IMPORTANT: build directories. In the example given, the "`configure`" script is run in the root of the SPELL source code directory. It is also possible, and usually desirable, to create a "`build`" directory inside the SPELL directory, go into it, and execute the "`configure`" script and the "`make`" program from there. The result of the configuration will be the same but all temporary files (configuration files and object files) generated during the build will appear inside the "`build`" directory, instead of being mixed with the source code files.

2.4 Compilation

In order to compile the selected components, the command "make" is to be executed in the build directory:

```
$> make
```

The make program will compile every selected SPELL component in the proper order, and will generate all needed libraries and program files.

2.5 Installation

In order to install the selected components, the command "make install" is to be executed in the build directory:

```
$> make install
```


The make program will copy and install all selected SPELL components, including the configuration and data files, sample procedures, and launcher scripts, into the desired target location.

After installation, the **SPELL_HOME** environment variable shall be defined and pointing to the SPELL installation base directory.

A typical example of the target location, once the installation process finishes can be seen below. In the example, it is assumed that **all** SPELL components have been enabled (thus they are compiled and installed):

bin	SPELL program files
config	Configuration files
data	Program input files
drivers	SPELL drivers
lib	Shared libraries
log	Log files directory
doc	Documentation directory
rcp	RCP libraries
server	Server application files
spel-gui	SPELL GUI application
spell	Language and library files

It is also possible to compile single components by moving to their build directory, and running "make" command inside.

03 February, 2015	SPELL Build Manual	
Page 11 of 16	File: SPELL - Build Manual - 2.4.4.docx	

For example, for compiling just the SPELL shared libraries:

```
$> cd workspace/lib  
$> make
```

Or to compile *and* install the `SPELL_UTIL` library:

```
$> cd workspace/lib/SPELL_UTIL  
$> make install
```

2.6 Cleanup

It may be desirable at some point to cleanup the files generated by the configuration and compilation. To do so, two commands can be used: "make clean" and "make distclean". The first one will delete all libraries, program files and object files created during the compilation. The second one will do the same thing, plus deleting all intermediate files generated by the "configure" script.

3 Building SPELL DEV

In this chapter, the steps for building and installing the SPELL development environment are described. It is assumed that the source code has been checked out from the repositories at the SPELL web site.

In the following, it is assumed that the source code is checked out in a directory named "workspace-dev".

3.1 Directory Structure

Below, the basic list of directories and files in the `workspace-dev` directory can be found:

<code>config</code>	<code>(spell)</code>	SPELL components configuration files
<code>configure.ac</code>	<code>(autotools)</code>	Main autotools configuration script
<code>data</code>	<code>(spell)</code>	SPELL components input files
<code>doc</code>	<code>(spell)</code>	SPELL documentation
<code>Makefile.am</code>	<code>(autotools)</code>	Makefile description input for autotools
<code>rcp</code>	<code>(spell)</code>	Eclipse RCP 3.5 libraries
<code>scripts</code>	<code>(spell)</code>	Scripts for launching some SPELL programs
<code>src</code>	<code>(spell)</code>	SPELL programs source files

This set of files and directories corresponds to a completely clean SPELL DEV source code package, as it comes in the source distribution. This package is still not prepared for build, since the auto-tools working files have not been generated yet. They are generated in the *preparation stage*, described in the next section.

3.2 Preparation

First the auto-tools working files have to be generated. This is done by running the "autoreconf" command in the base directory of the SPELL DEV source code:

```
$> cd workspace-dev
$> autoreconf -fi
```

This program will parse the `configure.ac` main script, and will create and install the standard files required for the compilation.

The list of files and directories after running `autoreconf` should be something like this (the new files and directories are highlighted):

aclocal.m4	(autotools)	Autoconf M4 macros
autom4te.cache	(autotools)	Cached data used by autoconf
config	(spell)	SPELL components configuration files
config.h.in	(autotools)	Input file for the standard config.h
config.sub	(autotools)	Utility script for autoconf
configure	(autotools)	Main build configuration script
configure.ac	(autotools)	Main autotools configuration script
data	(spell)	SPELL components input files
doc	(spell)	SPELL documentation
install-sh	(autotools)	Utility script for autoconf
Makefile.am	(autotools)	Makefile description input for autotools
Makefile.in	(autotools)	Intermediate makefile data
missing	(autotools)	Utility script for autoconf
rcp	(spell)	Eclipse RCP 3.5 libraries
scripts	(spell)	Scripts for launching some SPELL programs
src	(spell)	SPELL programs source files

Notice that additional "Makefile.in" files will be created everywhere in the directories where a "Makefile.am" file can be found. At this point, the source code is prepared to be configured with specific build options, and then compiled.

3.3 Configuration

Before compiling the source code, the build has to be configured in order to specify some options. These include the installation target directory, the location of some library dependencies and so forth.

To configure the build with specific options, the "configure" script generated by `autoreconf` is to be used. This script follows the standards for building and compiling any GNU software.

The "configure" script can show a description of all available options by running it with the "--help" argument:

```
$> ./configure --help
```

Part of the command output is presented below:

```
`configure' configures SPELL to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

...
  --with-eclipse=PATH    Provide eclipse installation used to compile RCP components
...
```

Only the parameters specific to SPELL (not GNU standard) have been shown.

Notice the mandatory "--with-eclipse" argument. The user shall provide the path to an existing Eclipse® IDE installation with this argument, otherwise the build process will fail.

Another configure parameter, this time part of the GNU standards, is "--prefix". This parameter can be used to specify the final location where the SPELL DEV application will be deployed after compilation. If this parameter is not given, the SPELL DEV application will be installed following the GNU standards.

IMPORTANT: as of the current revision, it is required to use the "--prefix" parameter to specify an installation directory for SPELL DEV. This way, all the SPELL DEV components will be installed together under the same location. If this parameter is not given, the software will be distributed in different directories of the operating system (/usr/bin, /usr/lib, etc.) and will break the SPELL development environment setup.

WARNING: you may need to specify the path to the Python shared library file. If that is the case, the configure script will warn you about it and tell you how to solve the problem.

Please refer to the "configure" script help for details about this parameter and others.

An example of a complete command to configure the build is shown below:

```
$> ./configure
      --prefix=/home/spell/SPELL-DEV
      --with-eclipse=/opt/eclipse-3.7
```

This command will build the components Executor, GUI, drivers, Shell and will use the Eclipse® installation located in "/opt/eclipse-3.7" directory.

As a result of the "configure" command, a set of "Makefile" files are generated. These make files are the final inputs for the GNU make program, that will perform the software compilation.

IMPORTANT: build directories. In the example given, the "configure" script is run in the root of the SPELL source code directory. It is also possible, and usually desirable, to create a "build" directory inside the SPELL directory, go into it, and execute the "configure" script and the "make" program from there. The result of the configuration will be the same but all temporary files (configuration files and object files) generated during the build will appear inside the "build" directory, instead of being mixed with the source code files.

3.4 Compilation

In order to compile the selected components, the command "make" is to be executed in the build directory:

```
$> make
```

The make program will compile every selected SPELL DEV component in the proper order, and will generate all needed libraries and program files.

3.5 Installation

In order to install the selected components, the command "make install" is to be executed in the build directory:


```
$> make install
```

The make program will copy and install all SPELL DEV components, including the configuration and data files and launcher scripts, into the desired target location.

A typical example of the target location, once the installation process finishes can be seen below.

bin	SPELL program files
config	Configuration files
data	Program input files
log	Log files directory
doc	Documentation directory
rcp	RCP libraries
spell-dev	SPELL DEV application

It is also possible to compile single components by moving to their build directory, and running "make" command inside.

03 February, 2015	SPELL Build Manual	
Page 16 of 16	File: SPELL - Build Manual - 2.4.4.docx	

3.6 Cleanup

It may be desirable at some point to cleanup the files generated by the configuration and compilation. To do so, two commands can be used: `"make clean"` and `"make distclean"`. The first one will delete all libraries, program files and object files created during the compilation. The second one will do the same thing, plus deleting all intermediate files generated by the `"configure"` script.