

SoftRelTools Manual

James Amundson
Computing Division
Fermi National Accelerator Laboratory

Manual Version 1.02
December 14, 1999

Contents

1	Introduction	4
2	SoftRelTools for users	6
2.1	Introduction	6
2.2	Details	7
3	SoftRelTools for librarians	7
3.1	Installing SoftRelTools	7
3.2	Setting project preferences	9
3.3	The project-specific package	9
3.4	The site-specific package	10
4	The inheritance hierarchy	10
4.1	Introduction	10
4.2	Details	11
4.2.1	Makefile fragments	11
4.2.2	Scripts	11
4.2.3	Examples	11
5	The environment variables	12
5.1	srt_setup and srt_environment	12
5.2	System defaults	14
5.3	User defaults	14
6	The code management system	14
6.1	Introduction	14
6.2	System preferences for newrel	15
6.3	User preferences for newrel	15
7	The build system	15
7.1	Exported headers	16
7.2	Subdirectories and Subpackages	16
7.2.1	Preferences for Subdirectories and Subpackages	17
7.3	Libraries	17
7.3.1	Preferences for Libraries	17
7.4	Binaries and Test Binaries	18
7.4.1	Scripts	18

7.4.2	Simple binaries	18
7.4.3	Complex binaries	18
7.4.4	Link libraries for simple and complex binaries	19
7.4.5	Rules for test binaries	19
7.4.6	Preferences for binaries	19
7.5	Standalone objects	19
7.6	Man pages	20
7.7	Documentation files	20
7.8	Generated include files	20
7.9	Generated code	20
7.10	SoftRelTools version	20
7.11	Building with qualifiers	21
7.11.1	Examples	21
7.12	Debugging	22
7.12.1	VERBOSE flag	22
7.12.2	echo_*	22
7.12.3	sortecho_*	22
7.12.4	gmake -debug	22
7.13	Example packages	22
8	The external package system	23
8.1	Standard compilers and linkers: arch_spec.mk	23
8.1.1	Internal structure of arch_spec.mk	23
8.1.2	Macros defined by arch_spec.mk	23
8.1.3	Qualifiers in arch_spec.mk	23
8.2	External libraries, etc.: arch_spec_*.mk	24
9	Availability	25

About this manual

This manual describes SoftRelTools version 2. It is intended to augment the documentation in “A UNIX Based Software Management System,” edited by Robert Harris, Computing Division Note: GU0013, which provides a more general introduction to SoftRelTools. It is available at

```
<http://www-  
cdf.fnal.gov/offline/code_management/run2_cmgt/run2_cmgt.html>.
```

Manual Changelog

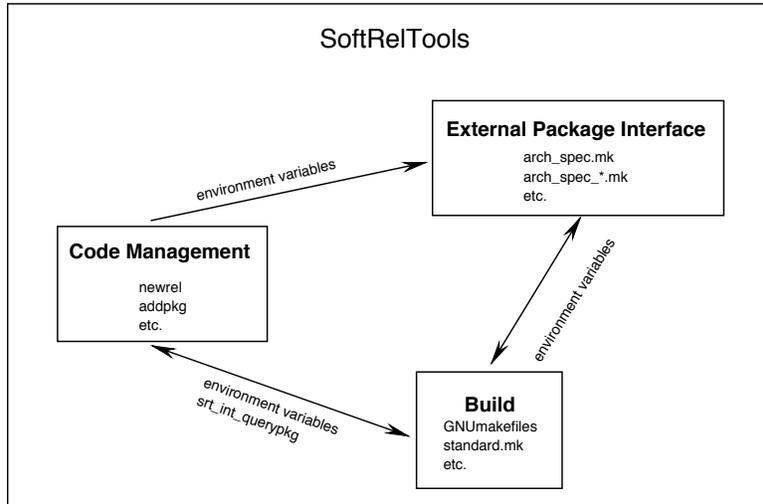
v1.01 April 28, 1999: Initial public version.

v1.02 December 14, 1999: Updated name of cvs repository.

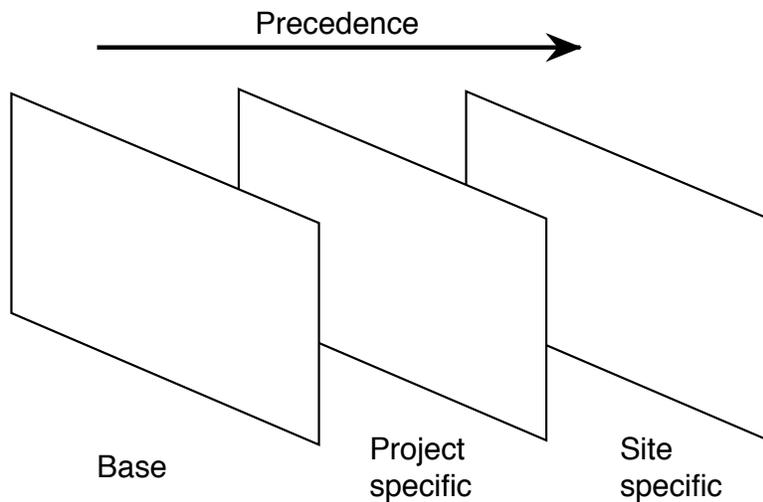
1 Introduction

The primary purpose of SoftRelTools version 2 is to provide a backward-compatible replacement for the original SoftRelTools written by Bob Jacobsen. SoftRelTools is designed to be easier to use and to maintain than the original SoftRelTools. Ease of use and maintainability are enhanced by increased functionality and modular structure. Maintainability is further enhanced by the separation of the basic tool from project and site specific settings through an inheritance hierarchy.

The modules and their relationships are described by the following figure:



The inheritance hierarchy from base to project and site-specific settings is orthogonal to the separation of the tool into modules. In the figure below, each pane represents the collection of modules shown in the figure above.



The SoftRelTools base is largely self-contained. The project and site specific behavior is implemented through optional additional packages, `SRT_$PROJECT` and `SRT_SITE`.

2 SoftRelTools for users

2.1 Introduction

All of the publicly available SoftRelTools files live in a distribution directory. For the purposes of the manual we refer to this directory as `$SRT_DIST`. In order to use SoftRelTools, users must have sourced the file `$SRT_DIST/srt/srt.csh` or `$SRT_DIST/srt/srt.sh`, depending on the users' shell. Of course, this assumes that a SoftRelTools distribution in `$SRT_DIST` has been installed. (See SoftRelTools for librarians.) After `srt.(c)sh` has been sourced, the command "`srt_setup`" will set up SoftRelTools for use. `srt_setup` is described in the `srt_setup` man page and in the section on the SoftRelTools environment.

The following example shows how a user can work with a package "Hello" that is part of a release "development". It assumes that SRT has been set up.

1. Create a test release called "myrelease"

```
newrel --test development myrelease
cd myrelease
```

2. Add the Hello package to "myrelease"

```
addpkg Hello
```

3. Set the local context, i.e., tell SRT to work with the current release. This is optional, but it speeds compilation.

```
srt_setup -a
```

4. Build all packages in the release

```
gmake
```

5. Here is a transcript of the steps above on my machine:

```
> newrel --test development myrelease
read user srtrc
Creating a test release "myrelease" in the directory
  /home/amundson/work
Linking tmp to /tmp/myrelease/tmp
```

```

Linking bin to /tmp/myrelease/bin
Linking lib to /tmp/myrelease/lib
> cd myrelease/
> addpkg Hello
Release development uses Hello ver-
sion HEAD, will check that out
Adding package "Hello" to ".".
cvs checkout: Updating Hello
U Hello/GNUMakefile
U Hello/Hello.cc
U Hello/Hello.h
U Hello/HelloExample.cc
> srt_setup -a
> gmake
< **include** >
< **include** > Hello
< **lib** >
< **lib** > Hello
< **compiling** > Hello.cc
< **building library** > libHello
< **bin** >
< **bin** > Hello
< **compiling** > HelloExample.cc
< **building** > HelloExample
>

```

The lines “read user srtc” and “Linking... to ...” after newrel are a result of my `~/srtc` file. See the section on user preferences for the code management system.

2.2 Details

To learn more about the commands `newrel`, `addpkg`, etc., read the section on the code management system. To learn more about building your own packages, read the section on the build system.

3 SoftRelTools for librarians

3.1 Installing SoftRelTools

(The following information is duplicated in the file `README.install` in the `SoftRelTools/install` directory.)

This version of SoftRelTools needs two things to get started:

1. A working boot release.
2. A properly installed srt directory in the main release area.

Once these two things are in place, users must

1. Source the `$$SRT_DIST/srt/srt.(c)sh` file appropriate for their shells.
2. Execute `"srt_setup"`.

Instructions for creating the boot release and srt directory follow:

To create a completely new SoftRelTools distribution:

1. Get `srt_distribution.tar` and untar it. (`srt_distribution.tar` is distributed separately.)
2. You now have a directory called `srt_distribution`. Move/rename it to whatever you want, but the instructions will refer to this directory as `srt_distribution`.
3. `cd` to `srt_distribution/srt` directory.
4. Execute the install script with `./install -p <your project>`. You can see the options to install with `./install -help`. The install script writes the location of the srt directory to the `srt.(c)sh` files. If you move the distribution, you will have to run install again. The project and cvsroot files can be modified by hand if you wish.

To update an old SoftRelTools distribution to work with the new SoftRelTools:

1. Untar the file `boot_release.tar` from this directory in your distribution "releases" directory.
2. Copy the srt directory from this directory to the distribution directory (the directory the old SoftRelTools called `$BFDIST`.)
3. `cd` to the new srt directory
4. See step 4above.

3.2 Setting project preferences

There are several new files that can hold settings that apply to all users.

\$SRT_DIST/srt/cvsroot holds the default value of CVSROOT for the distribution. It can be overridden for each package; see below. The contents of this file can be set by the install script or by hand.

\$SRT_DIST/srt/project holds the name of the project. It corresponds to the EXPERIMENT variable in the old SRT. There can be only one project per distribution. The contents of this file can be set by the install script or by hand.

\$SRT_DIST/srt/srt.csh is the file to be sources by csh and tcsh users. Only one line of it should ever be modified. The install script modifies it for you.

\$SRT_DIST/srt/srt.sh the file to be sources by ksh, zsh and bash users. Only one line of it should ever be modified. The install script modifies it for you.

\$SRT_DIST/srt/srt_envrc contains default environment variable settings. See the section on environment variables.

\$SRT_DIST/srt/srtrc contains default system settings for directory creation/linking in newrel. See the section on code management.

\$SRT_DIST/packages/<package>/cvsroot is an optional file for each package. If it exists, addpkg will automatically use it to determine the value of CVSROOT for <package>. This file will be written automatically if newpkg is called with the -d <cvsroot> argument. It can also be modified by hand.

3.3 The project-specific package

SoftRelTools will look for the package SRT_\$SRT_PROJECT (i.e., SRT_D0, SRT_CDF, etc.). This package can be used to augment and/or replace behavior of the central SoftRelTools. The mechanism for this is described in the section on the inheritance hierarchy.

3.4 The site-specific package

The package `SRT_SITE` (note the fixed name) is similar to the project-specific package. It is only intended for situations where two different machines for the same project need to behave differently. Its behavior supersedes both the default behavior and the project-specific behavior.

4 The inheritance hierarchy

4.1 Introduction

The version of `SoftRelTools` at Fermilab is modified frequently – on the order of once per day. These changes fall into two categories: 1) Bug fixes. 2) Local changes in settings, etc. The goal of the inheritance hierarchy in `SoftRelTools` is to allow projects to be able to make both kinds of changes quickly, without having to worry about affecting other projects. Changes of the first type can be incorporated into the base package in a controlled manner. Changes of the second type can stay with the projects, where they belong.

As the term implies, the inheritance hierarchy is based on the concepts of object-oriented design. Think of the site-specific parts of `SoftRelTools` inheriting from the project-specific parts, which inherit from the base. However, `SoftRelTools` is implemented primarily in GNU Make and Bourne Shell – neither of which are very well-suited to object-oriented programming. “Inheritance” in `SoftRelTools` should really be considered an analogy. The analogy will break down if pushed too far.

All of the modules in `SoftRelTools` can be modified first at the project level then the site level. The site has the final word. There are two mechanisms available:

1. All the modules of `SoftRelTools` look in the “special” subdirectory of the project and site packages. If a file corresponding to the current file exists in the “special” directory it is sourced. The specialization files are included in addition to the original file, the purpose being to augment or modify the behavior of the base package. This is the preferred method of incorporating changes.
2. The include paths in `SoftRelTools` also look in the project and site packages. This allows replacement of the `SoftRelTools` base files. The replacement

method should only be used if the override method is unsuitable. It is considered a goal that replacement does not need to be used. Replacement files go in the “SoftRelTools” subdirectory of the project and site packages.

4.2 Details

4.2.1 Makefile fragments

All makefile fragments in SoftRelTools/include look for the a corresponding file in SRT_\$\$SRT_PROJECT/special. Files in subdirectories of SoftRelTools/include go into the corresponding subdirectories of SRT_\$\$SRT_PROJECT/special. A few special files can be specialized both at the beginning and the end:

standard.mk looks for pre_standard.mk and post_standard.mk.

GNUmakefile.main looks for pre_GNUmakefile.main and post_GNUmakefile.main.

All other files look for files of the same name as themselves.

4.2.2 Scripts

All of the scripts in SoftRelTools consist of a set of subroutines, including one called “main”. If specialization files are found in the “scripts” subdirectory of the project-specific package, they are sourced before any of the subroutines are called. The specialization files can contain replacements for old subroutines and/or new subroutines.

4.2.3 Examples

- Add a new target to standard.mk

- put the lines

```
foo:
    echo "bar"
```

in special/post_standard.mk in the project-specific package. Now “gmake foo” will return “bar”.

- Add the `-no_exceptions` flag to the C++ compiler flags for the Kai compiler:

- put the line

```
override CXXFLAGS += --no_exceptions
```

in the special/compilers/KCC.mk in the project-specific package.

- Provide additional actions for addpkg

- put the lines

```
extra_actions () {  
    echo "Now executing top secret ex-  
tra actions"  
}  
  
main () {  
    script_defaults  
    process_args $*  
    actions  
    ods_actions  
}
```

in scripts/addpkg in the project-specific package. Now addpkg will print an extra message every time it is called. Notice that this example required both providing a new subroutine, `extra_actions`, and a replacing an existing routine, `main`.

5 The environment variables

5.1 srt_setup and srt_environment

SoftRelTools includes two commands, `srt_setup` and `srt_environment`, for examining and manipulating the user environmental variables. The former is actually an alias that calls the latter with certain options. The default behavior for `srt_environment` is to print the current settings. Sample output is below.

```
SRT settings:  
Variables for backward compatibility:  
BFARCH = Linux2-KCC_3_3
```

```
BFDIST = /home/amundson/work/dist
BFCURRENT = development
```

Automatic and derived variables:

```
PATH = /home/amundson/work/myrelease/bin/Linux2-
KCC_3_3:/home/amundson/work/dist
/releases/development/bin/Linux2-
KCC_3_3:/fnal/ups/prd/kai/v3_3f_1/Linux2/KCC_B
ASE/bin:/home/amundson/work/dist/releases/boot/bin/generic:/opt/kde/bin:/fnal/u
s/prd/ups/v4_3/Linux2/bin:/opt/kde/bin:/home/amundson/work/dist/releases/boot/
in/generic:/home/amundson/bin:/usr/sbin:/bin:/usr/bin:/etc:/usr/etc:/usr/bin/X1
:/usr/local/bin:./home/t1/amundson/bin:/home/t1/amundson/bin
LD_LIBRARY_PATH = /home/amundson/work/myrelease/lib/Linux2-
KCC_3_3:/home/amundso
n/work/dist/releases/development/lib/Linux2-KCC_3_3:
SRT_PRIVATE_CONTEXT = /home/amundson/work/myrelease
SRT_PUBLIC_CONTEXT = /home/amundson/work/dist/releases/development
MAKEFILES = "SoftRelTools/preamble.mk"
MAKEFLAGS = "-r -
I/home/amundson/work/myrelease/SRT_ODS -
I/home/amundson/work/di
st/releases/development/SRT_ODS -
I/home/amundson/work/myrelease/include -I/home/
amundson/work/dist/releases/development/include"
CVSROOT = /home/amundson/repository
SRT_SUBDIR = Linux2-KCC_3_3
SRT_PROJECT = ODS
SRT_ARCH = Linux2
SRT_ENV_SET = yes
```

User settable variables:

```
SRT_LOCAL = /home/amundson/work/myrelease
SRT_DIST = /home/amundson/work/dist
SRT_BASE_RELEASE = development
SRT_CXX = KCC_3_3
SRT_QUAL = default
```

The most important changes from the original SoftRelTools settings are the replacement of SRT_ for BF as the variable prefix and the splitting of the BFARCH architecture-C++ compiler combination into SRT_ARCH and SRT_CPP, respectively. SoftRelTools currently maintains the appropriate values of the BF variables, but it does not use them internally

The first time srt_setup is called, variables are set to their default settings, SRT_CXX=\$DEFAULT_SRT_CXX, etc. The defaults can be restored later by

“srt_setup -d”.

Users can alter the values of variables by putting one or more assignments on the command line

```
srt_setup SRT_CXX=EGCS_1_1
srt_setup SRT_QUAL=maxopt SRT_BASE_RELEASE=test
```

5.2 System defaults

srt_setup and srt_environment source the file \$SRT_DIST/srt/srt_envrc. This allows the librarian to set defaults for an entire project. For example, the srt_envrc might contain the lines

```
DEFAULT_SRT_CXX=KCC_3_3
DEFAULT_SRT_BASE_RELEASE=development
```

Since the file is sourced by the shell script, it can contain any shell commands. Only the resulting value of the variables matter.

5.3 User defaults

srt_setup and srt_environment also source the file \$HOME/.srt_envrc. (Note the presence of a leading dot in the user file, but not in the system file.) This allows the user to set his/her own defaults. Again, the file is sourced by the shell script, so it can contain any shell commands.

6 The code management system

6.1 Introduction

The scripts for the code management system are described by the man pages. Additionally, every script will describe its own actions and options when invoked with “-help” or an argument it does not understand. The “newrel” command looks for system and user preferences.

Scripts with the prefix “srt_int” are used internally by SoftRelTools. They will not generally be useful to users. Note that the srt_int_querypkg script is technically part of the build system, not the code management system. That means that changes to the build system can affect srt_int_querypkg. The code management scripts are designed to be independent of the build system.

6.2 System preferences for newrel

newrel sources the file `$SRT_DIST/srt/srtrc` for directory creation preferences. If the variable “extra_dirs” is defined, it is merged with the list of directories to be created. It should be in the form of a space-separated list of directory names. extra_dirs has two purposes:

1. Extra directories can be created.
2. Directories can be made into links to other areas. The syntax for this is “foo>/tmp/bar”, which means that the directory foo will be made into a link to /tmp/bar. If /tmp/bar does not exist, newrel will (attempt to) create it.

As in other places, srtrc is really a script which is sourced, so shell logic can be used. All that matters is the final value of extra_dirs. If “extra_dirs” is not defined, newrel looks for “stddirs” for backward compatibility with the old SoftRelTools.

The variable “release” (the name of the new release) is guaranteed to be available when srtrc is sourced.

6.3 User preferences for newrel

newrel also sources the file `$HOME/.srtrc` (note the leading dot) for directory creation preferences. See above for details. The following example .srtrc file is useful:

```
extra_dirs=\
"$extra_dirs tmp>/tmp/$release/tmp bin>/tmp/$release/bin lib>/tmp/$release/lib
```

It redirects all the directories containing large binary files to /tmp in such a way as not to interfere with other releases. Adding to the previous value of \$extra_dirs makes certain that system-level defaults are also respected.

7 The build system

Introduction

The build system in SoftRelTools is based on GNU Make. Make is a very flexible tool with a steep learning curve. SoftRelTools allows users to build and install a variety of objects without learning the intricacies of Make. At the same time, the

power of Make is available for users who need to go beyond simple functionality. SoftRelTools provides a method to build libraries, binaries, standalone object files, man pages and documentation files. It also provides a method for packages to use multiple subdirectories and subpackages. Packages make their header files available to other packages through a configurable directory.

In order to use the SoftRelTools build system, the user must create a GNU-makefile. The GNUmakefile must at least include the line

```
include SoftRelTools/standard.mk
```

at the end of the file.

7.1 Exported headers

All of the header files that are to be made available to other packages need to be placed in one subdirectory. The subdirectory may be the main package directory. SoftRelTools will use the subdirectory indicated by the variable `PACKAGE_INCLUDE`. If `PACKAGE_INCLUDE` is not found, SoftRelTools will look for a subdirectory with the same name as the package directory. If that fails, it will then look for subdirectories named `include`, then `src`, in that order. As a last resort, it will use the package directory itself as the header directory.

Exported headers can be used as follows

```
#include <Package/Header.h>
```

where `Package` is any package in the SoftRelTools distribution, including the current package. SoftRelTools sets the include paths accordingly.

7.2 Subdirectories and Subpackages

Packages can use an arbitrary hierarchy of subdirectories. SoftRelTools will attempt to launch builds in the subdirectories listed in the variable `SUBDIRS`. SoftRelTools will only launch make in directories containing a GNUmakefile. All subdirectories put their intermediate build products (object files, dependency files, etc.) in the same temporary directory.

Subpackages are very similar to subdirectories. A subpackage is distinguished by setting the variable `SUBPACKAGE` to the subpackage name. Each subdirectory of the subpackage must define `SUBPACKAGE`. Subpackages have their own temporary areas. Packages can have a mixture of subdirectories and subpackages, but this is not necessarily encouraged.

7.2.1 Preferences for Subdirectories and Subpackages

If the variable “SORT_SUBDIRS” is set, subdirectories are built in alphabetical order. Otherwise they are built in the given order.

7.3 Libraries

SoftRelTools can build static and shared libraries. To build the library libFoo, include the line

```
LIB=libFoo.a
```

libFoo will be static, shared or both, if the variable LIB_TYPE is “static”, “shared”, or “both”, respectively. (“Both” means generating two libraries, e.g., libFoo.a and libFoo.so.) The default is to create static libraries. The suffix on the filename is irrelevant – SoftRelTools will give the library the appropriate suffix depending on library type and platform. Static and shared libraries can also be built by setting the variables SHAREDLIB and STATICLIB, respectively. The value of LIB_TYPE is ignored for SHAREDLIB and STATICLIB.

The contents of the created libraries are defined by the following variables:

LIBCCFILES C++ files with the suffix .cc.

LIBCXXFILES C++ files with the suffix .cxx.

LIBCPPFILES C++ files with the suffix .cpp.

LIBCFILES C files with the suffix .c.

LIBFFILES Fortran files with suffixes .f or .F. The latter will be run through the C preprocessor before compiling.

LIBLIBS The contents of LIBLIBS are added to the end of the link line when the libraries are linked. They are not included in the dependencies.

7.3.1 Preferences for Libraries

As described above, libraries specified by the LIB variable will be built static, shared, or both depending on the value of the LIB_TYPE variable. The library rules in the old SoftRelTools included all the object files found in the library temporary directory into the library. It is preferable to link exactly those objects requested, however many packages rely on the old behavior. SoftRelTools will use

the old behavior (all objects in the directory) if the variable `CATCHALL_LIBS` is set. Otherwise, it will only link the requested objects.

7.4 Binaries and Test Binaries

SoftRelTools will attempt to build all files listed in the `BIN` variable during the bin stage. SoftRelTools will attempt to build all files listed in the `TBIN` variable during the `tbin` stage. The `tbin` stage is not normally built by “`gmake all`”; it must be invoked explicitly.

SoftRelTools provides rules for generating three different kinds of “binary” files.

7.4.1 Scripts

Files listed in the `SCRIPTS` variable are assumed to be scripts. They are copied into the binary destination directory and made executable. Note that normal usage requires listing the script in the `BIN` variable (to tell SoftRelTools that it needs to be built during the bin stage) *and* the `SCRIPT` variable (to tell SoftRelTools how to build it.) A single directory can build an arbitrary number of scripts.

7.4.2 Simple binaries

Each file listed in the `SIMPLEBINS` variable is built into a single binary of the same name. It looks for a single source file with the given name and a suffix `.cc`, `.cpp`, `.cxx`, `.c` or `.f`. They will each be linked with `BINLIBS`, described below. A single directory can build an arbitrary number of simple binaries. Note, however, that all simple binaries in a *package* are built in the same temporary directory. As with the other binary types, normal usage requires listing simple binaries in both the `BIN` variable and the `SIMPLEBINS` variable.

7.4.3 Complex binaries

A subdirectory may specify in `COMPLEXBIN` one complex binary to be built. The following contents may be specified:

BINCCFILES C++ files with the suffix `.cc`.

BINCXXFILES C++ files with the suffix `.cxx`.

BINCPPFILES C++ files with the suffix `.cpp`.

BINCFILES C files with the suffix `.c`.

BINFFILES Fortran files with suffixes `.f` or `.F`. The latter will be run through the C preprocessor before compiling.

BINSTANDALONEFILES Stand-alone object files to be linked with the binary.

The binary will be linked with **BINLIBS**, described below. Each complex binary has a unique temporary directory. As with the other binary types, normal usage requires listing simple binaries in both the **BIN** variable and the **COMPLEXBIN** variable.

7.4.4 Link libraries for simple and complex binaries

All binaries in a directory will be linked with the contents of **BINLIBS**. (Note that specifying extra libraries at link time is not usually a problem.) Local libraries should be listed as dependencies of the files that use them. Local libraries mean libraries built by **SoftRelTools**. External libraries, however, should *not* normally be listed as dependencies. A change in external libraries should be followed by a clean build. **SoftRelTools** places the contents of **BINLIBS** on the binary dependency line, but with the contents of **NODEP_LIBS** filtered out. Having libraries listed in **NODEP_LIBS** that are not in **BINLIBS** is perfectly acceptable. For backward compatibility, if **BINLIBS** is empty, **LOADLIBES** is used instead.

7.4.5 Rules for test binaries

All of the rules for binaries apply to test binaries, also.

7.4.6 Preferences for binaries

If **OLD_BIN_RULES** is defined, **SoftRelTools** includes the binary rules from the original **SoftRelTools**.

7.5 Standalone objects

Standalone objects are built directly in the library directory.

OBJCCFILES C++ files with the suffix `.cc`.

OBJCXXFILES C++ files with the suffix `.cxx`.

OBJCPPFILES C++ files with the suffix `.cpp`.

OBJCFILES C files with the suffix `.c`.

OBJFFILES Fortran files with suffixes `.f` or `.F`. The latter will be run through the C preprocessor before compiling.

7.6 Man pages

Files listed in the `MANPAGES` variable will be installed during the bin stage. `foo.1` will be installed in `man/man1`, `foo.2` will be installed in `man/man2` directory, etc.

7.7 Documentation files

Files listed in the `DOCS` variable will be installed into the `doc` directory during the bin stage.

7.8 Generated include files

Files listed in the `INC` variable will be built during the include stage. `SoftRelTools` does not provide the rules for building the files.

7.9 Generated code

Files listed in the `CODEGENFILES` variable will be built during the codegen files. `SoftRelTools` does not provide rules for code generation by default, but rules can be included by including the files `SoftRelTools/idl.mk`, `SoftRelTools/java.mk` and `SoftRelTools/yacc.mk`.

7.10 SoftRelTools version

`SoftRelTools` version 2 defines

```
SRT_VERSION=2
```

in case a package needs to check to see whether the rewrite is available.

7.11 Building with qualifiers

Qualifiers are named sets of flags. The name is placed in the `SRT_QUAL` environment variable. The value of `SRT_QUAL` passed to `srt_setup` determines which set of qualifiers are being used for the other packages. The local package can be built with any value of `SRT_QUAL`.

7.11.1 Examples

Building the hello package.

- A standard build

```
> srt_setup
> srt_setup -a
> gmake
< **compiling** > Hello.cc
< **building library** > libHello
< **building** > HelloExample
```

- A build using optimized settings

```
> srt_setup SRT_QUAL=maxopt
> gmake clean
> gmake
< **compiling** > Hello.cc
< **building library** > libHello
< **building** > HelloExample
```

- Use default release, but optimize locally by passing a different value of `SRT_QUAL` to `gmake`:

```
> srt_setup SRT_QUAL=default
> gmake clean
> gmake SRT_QUAL=maxopt
< **compiling** > Hello.cc
< **building library** > libHello
< **building** > HelloExample
```

Other combinations are possible. For example, a package that does not compile properly under optimization can easily redefine `SRT_QUAL` in the GNUmakefile itself, overriding other settings.

7.12 Debugging

SoftRelTools provides the following debugging aids:

7.12.1 VERBOSE flag

Normally, SoftRelTools executes its commands silently. If the variable `VERBOSE` is defined SoftRelTools will print each command before it is executed. The value of `VERBOSE` is irrelevant; it only has to be non-null.

7.12.2 echo_*

Typing `gmake echo_FOO` will echo the value of the variable `FOO` at target execution time. It also prints the value of the make “origin” command for the variable `FOO`. Obviously, it works for any variable.

7.12.3 sortecho_*

`sortecho_FOO` is similar to `echo_FOO`, but it sorts the contents of `FOO` and prints them in a single column suitable for input to `diff`. It is useful for comparing the values of complicated flags.

7.12.4 gmake -debug

This is really a function of `make`. The `debug` flag causes `make` to generate extremely verbose output. However, SoftRelTools works hard to minimize the extraneous content of the output.

7.13 Example packages

Several example packages are available:

Hello builds a hello world library and a corresponding executable.

BinExamples builds all the available binary types. It depends on `Hello`.

UsesSubpackages uses both subpackages and subdirectories.

TemplateArray is an example using C++ templates.

8 The external package system

The external package system breaks down into two parts: the interface to the standard compilers and linkers and the interface to external libraries, etc.

8.1 Standard compilers and linkers: `arch_spec.mk`

The external interface to `arch_spec.mk` is essentially unchanged: the build system includes `arch_spec.mk`, which sets a variables for the compilers, flags, etc. The most important change is that `SoftRelTools` relies on the `SRT_` variables as input instead of the `BF` variables. The `SoftRelTools` variables have been maintained.

8.1.1 Internal structure of `arch_spec.mk`

The original `SoftRelTools` contained all settings for all architecture/compiler combinations in one file. This has been substantially rearranged in `SoftRelTools` version 2. `arch_spec.mk` has been split into three: the main `arch_spec.mk`, the C++ compiler files and the platform files. The compiler and platform files live in the compiler and platform subdirectories of the include directory of `SoftRelTools`. Since some settings depend on both compiler and platform, a choice had to be made. The convention is that compiler-specific information lives in the compiler file, even if it is platform-specific. There are still `if` statements in the compiler files, but they are very simple.

8.1.2 Macros defined by `arch_spec.mk`

`SoftRelTools` now defines macros for compilation. All permutations of (C++, c, Fortran, Preprocessed Fortran) + (pic, non-pic)+ (with depends, without depends) are defined. Additionally, both on-the-fly and separate dependency generation are included. For details, see `arch_spec.mk`.

8.1.3 Qualifiers in `arch_spec.mk`

Qualifiers are named sets of flags. The name is placed in the `SRT_QUAL` environment variable. By default, `SoftRelTools` defines two sets:

default turns on debugging symbols and turns off optimization

maxopt turns off debugging symbols and sets the highest available generic optimization

Qualifiers have to be defined by hand in the compiler and platform files. There is no (nor can there be) any automatic way to define flags across architectures and compilers. New qualifiers can be defined in the project-specific package by specializing the compiler and platform files.

8.2 External libraries, etc.: `arch_spec_*.mk`

SoftRelTools attempts to create a standard for `arch_spec_*.mk` files where none existed before. `arch_spec_*.mk` files should:

- Follow the SoftRelTools inheritance hierarchy. This is achieved through the include path in make and the behavior of the file itself. The file's job is taken care of by including the file `override_arch_spec.mk`.
- Define a subset of the following:
 1. necessary C++ include path and/or C++ macro definitions
 - Modifies the variable `CPPFLAGS`
 2. necessary library path
 - Modifies the variable `LDFLAGS`
 3. necessary libraries
 - Modifies the variable `LOADLIBES`
 4. necessary executable path
 5. external dependencies
 - External dependencies are automatically included unless the variable `NO_AUTO_EXT_DEPENDS` is set.
 - The variable `arch_spec_depends` contains the external dependencies on exit.
- Optionally use an environmental variable if necessary to locate external packages.

- If the variable is not set, use a default value and set the variable `arch_spec_warning` to an appropriate warning string.
- If the package is unavailable set the variable `arch_spec_error` to an appropriate error string.
- Set the variable `arch_spec_warning` for other messages.

The error and warning messages are printed to stdout if the variable `VERBOSE` is set.

A few packages do not follow these guidelines for legacy reasons. Each one has a comment to that effect.

9 Availability

SoftRelTools is available for anonymous cvs access at

```
:pserver:anonymous@srtcvcs.fnal.gov:/srtcvcs
```

Login with

```
cvs -  
d :pserver:anonymous@srtcvcs.fnal.gov:/srtcvcs lo-  
gin
```

The password is “anoncvs”. The project-specific and example packages are available from the same location.

The SoftRelTools boot distribution is available at

```
<http://RunIIComputing.fnal.gov/cmgt/srt_distribution.tar.gz>.
```

This document is available at

```
<http://RunIIComputing.fnal.gov/cmgt/SoftRelTools-  
Manual/SoftRelTools-Manual.html>.
```

It is also part of the SoftRelTools package.