General SifDec documentation

N. I. M. Gould   D. Orban   Ph.L. Toint

September 16, 2004

Contents

Disclaimer

This software was written as a personal project and comes with NO WARRANTY of any kind, not even MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Please read the file LICENSE in the SifDec home directory prior to any other manipulation.

The authors assume no responsibility for any use.

The authors, N. I. M. Gould, D. Orban and Ph.L. Toint

Contact

N. I. M. Gould, Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, England.
n.gould@rl.ac.uk
http://www.cse.clrc.ac.uk/Person/N.I.M.Gould

D. Orban, École Polytechnique de Montréal
Mathematics and Industrial Engineering Department
Dominique.Orban@polymtl.ca
http://www.mgi.polymtl.ca/dominique.orban

Ph.L. Toint, Facultés Universitaires Notre-Dame de la Paix, 61, rue de Bruxelles, B-5000 Namur, Belgium.
Philippe.Toint@fundp.ac.be
http://www.fundp.ac.be/~phtoint

Note

This documentation is in constant evolution, and so is the software. We advise the reader to consult the website http://cuter.rl.ac.uk/cuter-www/sifdec for the latest information, bug fixes and patches concerning SifDec.

This document describes installation and usage of the SifDec package, and is intended to be one of the main documentation sources available with the package; other sources include man pages, various README files and self-documented scripts.

SifDec is a *decoder*. It translates test problems, written in Standard Input Format (SIF), into well-defined Fortran 77 and data files. Once translated, these files may be manipulated to provide tools suitable for testing optimization packages. SifDec used to be part of, and has been extensively used with, the CUTE testing environment [BCGT95] and is now a vital component of the CUTEr testing environment, which includes ready-to-use interfaces to existing packages, such as MINOS, SNOPT, filterSQP and KNITRO, but could also serve different purposes.

SifDec is now distributed as a separate package for both convenience reasons and to encourage consistent use in conjunction with other software and packages.

SifDec is available on a variety of UNIX platforms, including LINUX and is designed to be accessible and easily manageable on heterogeneous networks.

"When all else fails, read the documentation." (fortune)

Chapter 1

Installation and usage

1.1   Installing and managing SifDec

The current version of SifDec comes in the form of a gzipped tarfile. To uncompress and extract the SifDec distribution from it, move the file to a new directory of your choice—we shall refer to this directory as `$SIFDEC`—and issue the commands

```
prompt% gunzip sifdec.tar.gz
prompt% tar xvf sifdec.tar
```

or, more compactly,

```
prompt% gunzip -c sifdec.tar.gz | tar xvf -
```

On GNU-based LINUX systems, this is also done by the single command

```
prompt% tar zxvf sifdec.tar.gz
```

If you want the SifDec files to be accessible to a number of users on a shared filesystem on your local network, you might need privileged access to your machines, or to have these steps performed by your system administrator.

The current installation is via a text-based interface, in which the user is prompted for choices pertaining to the desired installation. The main installation script is `install_sifdec` and interacts with a number of auxiliary scripts. We examine these scripts in turn, using an example of a SifDec installation on a shared-filesystem network. The scripts provided are:

1. `install_sifdec`: installs a new instance of SifDec on the system,

2. `update_sifdec`: updates files in an installed instance of SifDec,

3. `uninstall_sifdec`: remove a particular instance of SifDec installation.

In addition the the three above scripts, we will also examine a manner to re-generate parts of SifDec, due to the modification of one or more files.

These scripts can be found in

```
$SIFDEC/build/scripts
```

Suppose, by way of example, that your local network contains the following machines (amongst others).

1. a SUN Ultra workstation running Solaris with an installed Sun Fortran 90 compiler, f90,

2. an intel-based personal computer running LINUX for which the Gnu Fortran 77 compiler, g77, is installed, and

3. a Compaq Alpha running Tru-64 for which the Compaq Fortran 77 compiler, f77, is available.

In the remainder of this documentation, we describe the role of the aforementioned scripts based on this example of network. Assume that you wish to install an instance of SifDec for each of these machines, according to Table 1.1:

Machine	Compiler	Size	Precision
SUN	f90	large	double
Intel	g77	medium	single
Compaq	f90	large	double

Table 1.1: A possible installation of SifDec on a shared-filesystem network. Size refers to the (maximum) dimension of the examples you wish to run, while Precision denotes the floating-point precision required.

SifDec relies on portable *Makefiles*, generated from architecture-independent *Umakefiles*[1].

1.1.1 `install_sifdec`

This script serves the dual purposes of installing the initial instance of SifDec on your system and of installing an additional instance, for a different architecture, where by *architecture*, we mean the combination machine–operating system–compiler–size–precision. Assume you wish to install all your different instances of SifDec in the directory `$SIFDEC = /usr/share/sifdec/`. Unpacking the SifDec distribution in the `$SIFDEC` directory and launching the initial installation, say for the SUN Ultra machine, is done by simply typing

```
prompt% install_sifdec
```

at the command prompt. However, before issuing the install command, we recommend that you check the files *system*.cf in `$SIFDEC/config`, where '*system*' represents your operating system, to make sure the commands there are correctly defined for your environment, and that the temporary directory is correctly set. The current directory '.' must *not* be used as temporary directory. Once you have issued the `install_sifdec` command, you will be prompted for information regarding the instance of SifDec you wish to install. The first question concerns your machine. In this case, select "Sun workstation" (7). Next, select the operating system your machine is running. Here, we select "Solaris" (1). You are then faced with a list of available compilers for your machine (without any guarantee that these compilers are actually *installed* on your machine, simply those we know are available for the machine–operating system

---

[1]Umakefiles are a much simplified variant of Imakefiles [Dub83]

combination you have selected). We want to select "Sun f90" (8). Select next the precision of the SifDec tools (<u>s</u>ingle or <u>d</u>ouble), and their size (<u>s</u>mall, <u>m</u>edium, <u>l</u>arge or <u>c</u>ustomized).

Once this information has been provided to the installation script, you are given a default directory name where the selected instance of SifDec will be installed. This directory is a subdirectory of `$SIFDEC` that you chose earlier (in this case, `/usr/share/sifdec/`). For the present instance, the default directory is

> `/usr/share/sifdec/SifDec.large.sun.sol.f90`

reflecting the selections you made during the early installation phase. This directory name should be self-explanatory and should help you and other users determine where each installed instance of SifDec is actually stored. Notice that the precision is not reflected in the directory name. The reason is that both single and double precision instances of SifDec may be installed for the same machine-operating system-compiler-size combination; these will be stored in the `single/` and `double/` subdirectories of the above directory. If you wish, you may redefine the directory name and give it whatever name you like—it need not be a subdirectory of `$SIFDEC`. Note however that you should give the *full pathname* of the new directory that you choose, *e.g.*

> `/home/mjdpowell/software/yetAnotherSifdec`

even if this new directory is a subdirectory of the `$SIFDEC` directory:

> `/usr/share/sifdec/aCustomSifdec`

It is probably good practice to be content with the default name or not to give it a cryptic or ambiguous name. After checks to see if a similar distribution has already been installed and for the existence of the specified directory, the installation itself begins.

The script `install_sifdec` then creates the necessary directory structure, *Umakefiles* and configuration files. The final step of the installation is left to the user and is described below.

Once this phase is complete, `install_sifdec` reminds you of what you should add to your `.cshrc`, `.bashrc`, or whichever UNIX configuration file corresponds to the shell you use. The directory structure after the initial installation is as described in the CUTEr/SifDec paper provided in the SifDec distribution and in §1.2 and Fig.1.1. In the case we are concerned with, the `SIFDEC` environment variable should be set to `/usr/share/sifdec` and `MYSIFDEC` to `/usr/share/sifdec/SifDec.large.sun.sol.f90` (or the alternative directory you specified during the installation phase).

The script `install_sifdec` also advises you to read the various `README` files scattered over the directory tree under `$MYSIFDEC`. We now describe the final step of the installation using *Umakefiles*. There is a *Umakefile* in each subdirectory of `$MYSIFDEC`. Each of these *Umakefiles* needs to appropriately use the configuration files stored in `$MYSIFDEC/config` so as to generate *Makefiles* suited to your local system. This process is usually referred to as *bootstrapping*. This is done by changing to `$MYSIFDEC` and issuing the command

> `prompt% ./install_mysifdec`

at the command prompt. Please note that if both single and double precision were installed, the script `install_mysifdec` requires a command-line argument, telling it for which precision it should bootstrap the *Umakefiles*. In an attempt to follow the main guidelines for the CPP — the C preprocessor, on which *Umakefiles* are based — the argument to `install_mysifdec` takes the form of a symbol definition. More precisely, if the user wishes to remake the double precision version of SifDec, the command is

```
prompt% ./install_mysifdec -DDoublePrecision
```

and similarly, for the single precision version,

```
prompt% ./install_mysifdec -DSinglePrecision
```

Refer to the file `IMPORTANT` for the latest details. Do not let `make`'s output confuse you. On a Linux system, and because `make` is usually accompanied by the `-w` command-line option, using the standard g77 compiler, the output of the above command looks like

```
umake -I./config -DIsg77 -DLargeSize -DDoublePrecision
+ /bin/rm -f Makefile.bak
+ /bin/mv Makefile Makefile.bak
umake -I./config  -DTOPDIR=. -DCURDIR=.  -DIsg77 -DLargeSize -DDoublePrecision
making Makefiles in bin...
make[1]: Entering directory '/home/do/Sifdec4Linux/bin'
make[1]: Nothing to be done for 'Makefiles'.
make[1]: Leaving directory '/home/do/Sifdec4Linux/bin'
making Makefiles in double...
make[1]: Entering directory '/home/do/Sifdec4Linux/double'
making Makefiles in double/bin...
make[2]: Entering directory '/home/do/Sifdec4Linux/double/bin'
make[2]: Nothing to be done for 'Makefiles'.
make[2]: Leaving directory '/home/do/Sifdec4Linux/double/bin'
making Makefiles in double/config...
make[2]: Entering directory '/home/do/Sifdec4Linux/double/config'
make[2]: Nothing to be done for 'Makefiles'.
make[2]: Leaving directory '/home/do/Sifdec4Linux/double/config'
make[1]: Leaving directory '/home/do/Sifdec4Linux/double'
```

This is *normal* output and it indicates that everything worked out smoothly. `make` is simply echoing what it attempts to do in each subdirectory. A message like "Nothing to be done for 'Makefiles'." simply indicates that the subdirectory where `make` is currently working does not have further subdirectories. On most systems, `make` is less verbose.

The above command should be able to properly generate the *Makefiles* in each subdirectory. These *Makefiles* should also only contain standard commands, as every effort has been made to avoid using exotic *Makefile* features and capabilities, such as the `$@@` construct. A `README` file accompanies every *Makefile* to describe what it does and which targets it recognizes. Users are advised to take a look at these files. You normally do not need to read *Makefiles* generated by *umake* as these are usually very long and contain hundreds of parameter definitions. The documentation files and a basic knowledge of `make` should be enough for you to feel comfortable with the (re)generation of the various parts of SifDec. Once the *Makefiles* are generated, the only thing that remains to be done is the usual `make all`. However, as users who have some experience with `make` know, `make` outputs a lot of information—it basically echoes to the standard output every action it takes. The `-s` command-line option to `make` lowers its verbosity level and basic information on how the build is progressing only is printed. Thus, users should build SifDec using the command

```
prompt% make -s all
```

This command completes the installation of SifDec, using *Umakefiles*. On my Linux system, the installation takes a couple of minutes and `make`'s output looks like

```
 Getting UNIX commands right          [Ok]
 Casting       script.sed            [Ok]
 Casting       cast.sed             [Ok]
 Casting       cast90.sed           [Ok]
 Casting       local.f              [Ok]
 Adding        timer                [Ok]
 Building      local.o              [Ok]
making all in ./bin...
 Linking       slct                 [Ok]
 Linking       clsf                 [Ok]
 Casting       classall             [Ok]
 Casting       classify             [Ok]
 Casting       helpmsg              [Ok]
 Casting       select               [Ok]
 Casting       sifdecode            [Ok]
 Installing    show.awk             [Ok]
 Installing    param.awk            [Ok]
making all in ./double...
making all in double/bin...
 Building      sifdec.o             [Ok]
 Building      decode.o             [Ok]
 Building      gps.o                [Ok]
 Building      inlanc.o             [Ok]
 Building      makefn.o             [Ok]
 Building      mafnad.o             [Ok]
 Building      makegr.o             [Ok]
 Building      magrad.o             [Ok]
 Building      printp.o             [Ok]
 Building      rest.o               [Ok]
 Building      trans.o              [Ok]
 Building      utils.o              [Ok]
 Linking       sifdec               [Ok]
making all in double/config...
```

On workstations, the installation should be expected to take longer. During this phase, keep an eye on the screen and look for the `[Ok]` indicators. Should `make` come across some difficulty, this sequence of indicators should be interrupted by an error message. To know more about the problem, read the README file in the directory where the problem occured to try to indentify the target which `make` was attempting to build, and re-run `make` on that target without the `-s` option.

You may then install a new instance of SifDec, which may be for a different architecture, or one corresponding to an already-installed instance, with a different precision or size. In all cases, the environment variable `MYSIFDEC` should point to the current, working, instance of SifDec.

The `install_sifdec` script keeps track of all installed instances of SifDec on your system in the log-file `$SIFDEC/log/install.log`. This file may be used, for instance, to have `MYSIFDEC` point to the right distribution. For the purpose of illustrating the above, assume the three distributions given in Table 1.1 are installed in their default directory. Besides date information, the following will be found in `$SIFDEC/log/install.log`, where the exclamation mark (`!`) is a separator.

```
double large Sun-workstation sol f90   !  $SIFDEC/SifDec.large.sun.sol.f90
double medium Intel-like-PC lnx g77    !  $SIFDEC/SifDec.medium.pc.lnx.g77
double large Compaq-Alpha t64 f90      !  $SIFDEC/SifDec.large.alp.t64.f90
```

1.1.2 `update_sifdec`

As it is our intention to upgrade over time (or fix if necessary) the tools supplied in the SifDec package, a mechanism for keeping an installed system up to date, and to install newer instances of the tools, is required. This is the role of the `update_sifdec` script. If all goes well, you should not have to use `update_sifdec` immediately. Announcements of bug-fixes and enhancements will be posted and indicated on the website. There are two forms of the command.

In its first form, `update_sifdec` takes two command-line options, as follows

```
prompt% update_sifdec filename
```

where *filename* is the name of the file to upgrade, possibly specified with a path. Suppose, for example, that the file `gps.f` has been improved so as to perform its task faster, upgrading your current instance of SifDec is achieved by typing

```
prompt% update_sifdec gps.f
```

at the command prompt. This command first copies the new source file to proper location, which is in this case `$SIFDEC/common/src/tools`. If there are currently both single and double precision instances, you will be asked to choose which you would like to update; if there is only one instance under `$MYSIFDEC`, the precision will be chosen accordingly. The script then casts and compiles the incoming file. Of course, corresponding actions are performed depending on the type of *filename*: if it is a script, it is only cast, and stored in its proper place, and if it is a documentation file, it is simply moved to `$SIFDEC/common/doc`.

In its second form, `update_sifdec` takes three command-line options, described as follows

```
prompt% update_sifdec -a filename
```

where *filename* is the name of a file describing a list of SifDec files to be upgraded. The file *filename* should contain

1. on its first line, the directory where the new (upgraded) files can be found, and

2. on subsequent lines, the names of those upgraded files, possibly preceded by their destination directory. A single file per line should be given.

Note that preceding the file names by their destination directory is not compulsory; in fact, the path is ignored and `update_sifdec` tries to determine the correct path for itself. As an example, suppose that the tools `gps.f`, `install_sifdec`, `compiler.cry.unc.f90` and `sifdecode.pro` have been upgraded, and are temporarily stored in `/home/upgrade`. A corresponding input file might be

```
/home/upgrade
$SIFDEC/common/src/tools/gps.f
$SIFDEC/build/scripts/install_sifdec
compiler.cry.unc.f90
sifdecode.pro
```

but exactly the same result would be produced by the simpler file

```
/home/upgrade
gps.f
install_sifdec
compiler.cry.unc.f90
sifdecode.pro
```

or by the deliberately confusing file

```
/home/upgrade
/usr/share/junk/gps.f
/home/upgrade/install_sifdec
/home/downgrade/compiler.cry.unc.f90
/opt/degrade/sifdecode.pro
```

As above, SifDec copies these files from /home/upgrade to their proper location, prompts for the precision required (if necessary), casts and, where necessary, compiles the incoming files, and updates the specified instance stored under $MYSIFDEC.

The additional command-line option –m forces update_sifdec to simply move the files to their proper location and to skip compilation. Help may be obtained from update_sifdec through either of the –h, –help or --help flags.

To summarize, the complete synopsis of update_sifdec is as follows

```
update_sifdec [-h | -help | --help] [-m] [-a listFile | newFile]
```

In the situation where SifDec has been unpacked but no further installation steps were performed, or all current instances were deleted, update_sifdec still can move the updated source files to their proper location, skipping the compilation phase. The same syntax as above can be used.

Caution: attention should be paid to the fact that update_sifdec works by source-ing the UNIX commands from the file $MYSIFDEC/*precision*/config/cmds (where *precision* is the required precision) and that these commands define the temporary directory used during compilation phase. In most cases, this temporary directory is simply /tmp. This temporary directory *must not* be the same as that specified in the first line of update_sifdec's input file (/home/upgrade in the examples above). When no compilation occurs and update_sifdec simply moves the files to their proper location, it will source the file $SIFDEC/build/arch/system.all, a symbolic link to one of the other system.* files, rather than the command file, as the location of this will not yet have been assigned. The user might reset this link to better suit their system, or create a suitable system.all file of their own, better able to give the correct commands in all, or most, cases.

Please remark that it should be safer to update files using the –m command-line option to update_sifdec, and then to issue a make all (or make -s all) from $MYSIFDEC (or from the home directory of the SifDec instance which should be rebuilt).

1.1.3 `uninstall_sifdec`

The script `uninstall_sifdec` is used to remove a previously installed instance of SifDec from your system. If called with no argument, the user is asked to choose which distribution to remove, from a list of all the instances found on the system. Otherwise, the only argument is the name of the directory containing the distribution to be removed. We illustrate the second case. Referring again to Table 1.1, assume we wish to remove the Compaq-Alpha distribution. This is done by issuing the command

```
prompt% uninstall_sifdec $SIFDEC/SifDec.large.alp.t64.f90
```

at the command prompt. If this directory contains both the single and double precision instances, you will be prompted for which should be removed. There is no possibility, at the moment, to remove both instances at once. If single or double precision instance only is present, the whole directory will be deleted as will the corresponding entry in `$SIFDEC/log/install.log`. Note that un-installing should be done from the same machine from which the installation command was issued, as the corresponding directory might not be recognized on other machines. Issuing the command

```
prompt% uninstall_sifdec --help
```

will display a short help message. The script is itself self-documented and the user may consult it for more information.

1.1.4 Rebuilding SifDec

A rebuild of SifDec may turn out to be necessary whenever SifDec informs the user that the workspace dimensions need to be increased—a rebuild may also turn out to be necessary whenever prototype files are modified, or in general, whenever *any* basic file is modified. SifDec itself usually issues warning messages whenever the workspace is insufficient, urging the user to increase a particular (set of) parameters. These parameters may be tuned in `sifdec.siz`, which can be found in

```
$MYSIFDEC/precision/config
```

where *precision* is either 'single' or 'double', according to your installation. For the change to take effect, SifDec needs to be cast and compiled again. Assume the Solaris installation is modified.

All the user needs to do to make sure he or she rebuilds everything that needs to be rebuilt is change to the directory `$MYSIFDEC` and issue a

```
prompt% make -s all
```

`make` then takes care of everything and rebuilds whichever targets depend on the updated files.

1.2 The SifDec tree

One of the defects of CUTE is that it was not designed to simultaneously support a multi-platform environment, that is instances of the environment that could be used simultaneously from a central server on several (possibly different) machines at the same time. Moreover, using CUTE on a single machine in conjunction with several different compilers (a case that frequently occurs when testing new software) is impossible. Furthermore, handling different instances of the environment corresponding to different *sizes* of the tools (that is the size of the test problems that they can handle) is also impossible. The reason for

these difficulties is that the structure of the CUTE files, as described in [BCGT95], does not lend itself to such use, since it only contains a single subtree of objects files. If we call the combination of a machine, operating system, compiler and size of the tools an *architecture*, the obvious solution is then to allow several such subtrees in the installation, one for each architecture used.

However, as soon as the possibility of using architecture dependent subtrees is raised, the proper identification of the parts (scripts, programs) of the environment that are independent of the architecture also become an issue. Since it would be inefficient to store copies of these independent scripts and programs in each subtree, it is natural to store them in a data structure which is itself disjoint from the dependent subtrees. Finally, the multiplication of subtrees containing sometimes very similar but yet vitally different data makes the maintenance of the environment substantially more complicated, and therefore requires enhanced tools and a clear distinction between the parts of the environment that are related to decoding SIF-encoded files and those related to its own maintenance.

The directory organization chosen for SifDec, shown in Figure 1.1, reflects these preoccupations. We now briefly described its components.

Starting from the top of the figure, the first subtree under the main $SIFDEC directory (the main root of the SifDec environment) is build, which essentially contains all the files necessary for installation and maintenance. Its arch subdirectory contains the files defining all possible architectures that are supported by SifDec, allowing the user to install new architecture dependent subtrees in an evolving manner, depending on the testing needs, the evolution of the platforms, systems and compilers. The prototypes subdirectory contains the parts of the environment which have to be specialized to one architecture before it can be used. We call such files *prototypes* and the process of specializing them to a specific architecture *casting*. The prototype files include a number of tools and scripts whose final form typically depends on compiler options and the chosen size of the tools. Finally, the last subdirectory of build, named scripts, contains the environment maintenance tools as well as a number of documentation files.

The second subdirectory under $SIFDEC is called config and contains all the configuration and rules files which are relevant to *umake* when the latter is used to *bootstrap* the various *Imakfiles* in order to create the necessary *Makefiles*.

The third subtree under $SIFDEC is called common and contains the environment data files that are relevant for its purpose, but that are independent of the architecture. Its first subdirectory, doc, contains a number of documentation files concerning the environment (such as a description of its structure, the description of procedure to follow to fully exploit the possibilites of the SIF decoder, the complete SIF reference document, . . .), but not a description of the SifDec tools and scripts themselves. These are documented in the man subdirectory (and, as is common on Unix systems, its man1 subdirectory). The src subdirectory contains two subdirectories that contain the source files for the environment utilities: select contains the source of the classification and selection tools and sifdec contains the source of the SIF decoder.

The log subdirectory of $SIFDEC contains a log of the various installations (and, possibly, subsequent un-installations) of the environment for the various architectures.

The remaining subdirectories of $SIFDEC are all architecture dependent: each of them corresponds to the installation of SifDec on a specific machine, for a given operating system and compiler and for a given size. The figure only represents one, but the continuation dots at the bottom of the leftmost vertical line indicate that there might be more than one. The name of these directories are (by default) automatically chosen at installation, but a user of one of these subtrees would typically give it a symbolic name, like $MYSIFDEC, to refer to the instance of SifDec currently in use. Each architecture-dependent subtree is divided into its single and double precision instances (single and double, respectively), each of these containing
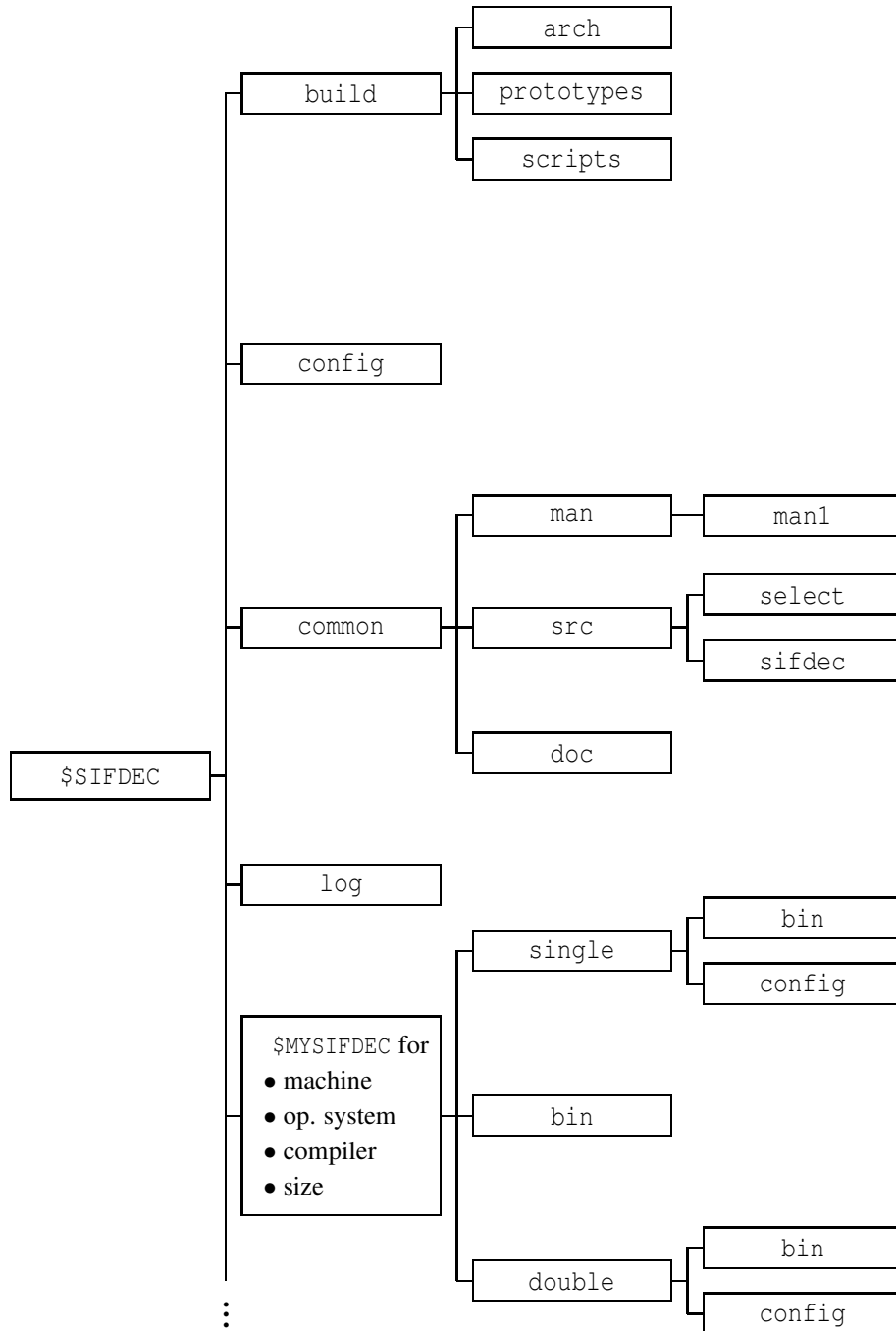
Figure 1.1: Structure of the SifDec directories

in turn two subdirectories. The first, `bin`, contains the `SIF` decoder executable in the corresponding precision. The second, `config`, contains the architecture dependent files that were used to build the current `$MYSIFDEC` subtree (they are reused when a tool or optimization package is added or updated). Finally, `$MYSIFDEC/bin` contains those scripts which are architecture-dependent, but not precision-dependent.

A final new feature of the environment organization is that the documentation is available via the usual `man` command for the scripts and tools, and both in acsii and pdf formats for the rest. It is hoped that this will make access to the relevant information more convenient for users.

1.3 User-modifiable parts

The SIF decoder depends on user-modifiable variables, which are not directly included in the Fortran source code, but cast prior to compilation. The file containing the user-modifiable data is `sifdec.siz`. After the initial installation, this file will be in the SifDec directory `$MYSIFDEC/[single|double]/config`. If modified, the SifDec distribution may be rebuilt using the new parameter values using `make`.

Some Fortran source files, like `clsf.f` and `slct.f`, have hardcoded user-modifiable parts. These are usually located at the top of the file, between banners, such as

```
C--------- THE FOLLOWING SPECIFICATIONS MAY BE MODIFIED BY THE USER ----------
```

and

```
C--------------- END OF THE USER MODIFIABLE SPECIFICATION -------------------
```

1.4 SifDec **sizes**

The SifDec package is distributed with three default "sizes": large, medium, and small. In addition, there is a *custom* size, which, as the term indicates, may be suitable for situations demanding a specialized configuration. These sizes refer to the size of the memory available for problem decoding and solution, and hence are directly related to the size (the amount of data) of the problems that SifDec can tackle. It may happen that the predetermined sizes do not fit your favorite problem or your machine, and that you wish to specify your own. Typically, when running too large a problem, SifDec will complain that one of the size parameters is too small and stop. You then have to increase this parameter (if this is possible on your machine) in order to handle the problem. This modification of the SifDec array sizes is explained below.

The actual choice of one of the predetermined sizes is made when running the `install_sifdec` command, which prompts the user for the desired size. In fact, `install_sifdec`, or the *Makefiles*, depending on which SifDec you are using, cast the source code against a "size mask" corresponding to the selected size, and thereby determines the dimensions of the various arrays used in the code. The assignment statements are differentiated by their first four characters:

`CBIG`	specifies the large size
`CMED`	the medium size
`CTOY`	the small size
`CCUS`	the custom size

Note that the custom size is first thought of as larger than the large size, but nothing prevents the user from building an intermediary size or a smaller size than the toy size.

Changing the size of the SifDec distribution in the sense just described may call for a partial re-installation. If most (or all) parameters must be, say, increased, it might be worth considering simply re-installing SifDec using a larger size (*e.g.* large if medium turns out to be insufficient for your purposes). To that end, execute install_sifdec again and select the correct size. In case very few parameters need to be changed, the procedure described below might be considered. We now examine this procedure in more detail.

1.4.1 `SIF` **decoder sizes**

The `SIF` decoder sizes are gathered in the file $MYSIFDEC/*precision*/config/sifdec.siz, which contains the following parameters.

Parameter	Brief description
NMAX	the maximum number of variables in a problem
NGMAX	the maximum number of groups in a problem
NGRMAX	the maximum number of different group types
NGPVMX	the maximum number of real parameters associated with groups
NELMAX	the maximum number of nonlinear element functions in a problem
NLMAX	the maximum number of different nonlinear element types
NEVMAX	the maximum total number of elemental variables in a problem
NINMAX	the maximum total number of internal variables in a problem
NSETVC	the maximum number of nonzero entries in an element Hessian
NEPVMX	the maximum number of real parameters associated with nonlinear elements
LA	the maximum number of nonzero entries in the linear elements
NINDEX	the maximum number of integer parameters in a problem
NRLNDX	the maximum number of real parameters in a problem
NBMAX	the maximum number of different sets of bounds on the variables which may be specified
NSMAX	the maximum number of different sets of starting points which may be specified
NOBMAX	the maximum number of different sets of bounds on the objective function which may be specified

Definitions of the terms used here are given in [CGT92]. These parameters are assigned a value upon initial installation of SifDec on the system. These values should be changed according to the `SIF`-decoder messages issued at decode-time, and SifDec should be rebuilt.

Changing compiler flags

In some circumstances, it might be useful to alter the predefined compiler flags. An example might be when some new level of code optimization becomes available on your machine. Note that care should be exercised with code optimizers: we know of cases where the optimizers introduce real bugs into the code. As a consequence, it might be a good idea to turn optimization off before deciding that some

strange behaviour of the package is anomalous and worth reporting. This is another reason why modifying compiler flags might be useful. Some operating system revisions might also require that you change machine dependent constants or procedures (such as the timer).

If compiler flags should be changed prior to a rebuild, the user should do so by altering the `COMPILE` and `LOAD` variables in the file $MYSIFDEC/*precision*/config/cmds.

If compiler flags should be changed to affect *all* subsequent installations of SifDec, the file `<your_system>.cf` should be modified accordingly, where the compiler in question is descibed. For instance, if the compiler in question is only found on SUN machines, the file `sun.cf` should be modified. If it may be found on any machine, the file `all.cf` should be modified. These files are found under $MYSIFDEC/config.

System dependent constants and functions

All the system dependent constants and functions are specified in the configuration files `all.cf` and/or `<your_system>.cf`, found under $MYSIFDEC/config, and also in the Fortran file $MYSIFDEC/*precision*/config/local.f after the initial installation. If these need to be changed, this latter file is the one on which to operate before rebuilding SifDec. Keep in mind that altering some or all these files will affect *all* subsequent installations of SifDec.

A set of hashing routines

The routines `HASHA`, `HASHB`, `HASHC` and `HASHE` provide a Fortran hashing tool. They are system dependent in that they rely on the number of bytes used to represent an integer within the particular Fortran dialect used. This number of bytes is set in the parameter `NBYTES` in the configuration files `all.cf` and/or `<your_system>.cf`, found under $MYSIFDEC/config. If your Fortran compiler uses an "unorthodox" number of bytes for its integers, you will have to change the value of `NBYTES`.

A definition of the arithmetic constants

The supplied functions `SMACHR` and `DMACHR` return values for various machine dependent constants, for single and double precision arithmetic, respectively. These machine constants are denoted Rn in single precision and Dn in double precision. We recapitulate them in the following table

Parameter	Brief description
R1, D1	the smallest positive number ε_1 such that $1 + \varepsilon_1 > 1$
R2, D2	the smallest positive number ε_2 such that $1 - \varepsilon_2 < 1$
R3, D3	the smallest nonzero positive number
R4, D4	the smallest full precision positive number;
R5, D5	the largest finite positive number

Each of these numbers should be modified, in `all.cf` and/or `<your_system>.cf`, found under $MYSIFDEC/config.

A CPU timer

This is a real function CPUTIM, that returns the current CPU-time used by the package, expressed in seconds. This timer is, unfortunately, highly system dependent. The specific code for CPUTIM is originally located in $SIFDEC/build/arch/compiler.*.*.* and concatenated with local.f at cast time, during installation.

1.4.2 Rebuilding SifDec

Rebuilding SifDec is done as described in §1.1.4. Simply change to $MYSIFDEC and issue a make -s all to make sure that everything that needs to be rebuilt is rebuilt.

1.5 The sifdecode command

It may be useful in some cases to decode a SIF-encoded problem without running an optimization package afterwards, or to simply check the syntax of the SIF file. Either in this respect, or as part of a bigger project—for instance, SifDec—the sifdecode command is invoked whenever it comes to decoding a SIF file. The sifdecode command recognizes a number of command-line options, many of which only make sense when sifdecode is used in conjunction with an optimization package. For more information on how to use SifDec in conjunction with CUTEr, refer to the manual [GOT01]. The options are described below.

```
sifdecode   [-s] [-h] [-k] [-o j] [-l secs] [-f] [-b] [-a j] [-show]
            [-param name=value[,name=value...]]  [-force] [-debug] probname[.SIF]
```

where

-s use the single-precision SIF decoder, if installed. This is useful to subsequently solve the problem in single precision, using some optimization package;

-h (or **[–help]**) prints a help message;

-k keep the load module after use; this is only useful in conjunction with some optimization package;

-r discourage recompilation of the test problem; this is only useful in conjunction with some optimization package;

-o *j* verbosity level: -o 0 is silent mode and -o 1 is verbose mode. The default is -o 0;

-l *secs* limits the CPU running time to *secs* seconds; this is only useful in conjunction with some optimization package;

-f generate the relevant subroutines for automatic differentiation in *forward* mode;

-b generate the relevant subroutines for automatic differentiation in *backward* mode;

-a *j* when used in conjunction with **-f** or **-b**, **-a 1** uses the older HSL automatic differentiation package AD01, which **-a 2** uses the newer, threadsafe, automatic differentiation package AD02;

-show displays possible parameter settings for probname[.SIF]. Other options are ignored;

-param casts probname[.SIF] against explicit parameter settings;

-debug links the libraries and compile with -g option so as to allow debugging; this is only useful in conjunction with some optimization package;

-n use the load module if it exists. The default is to recompile; this is only useful in conjunction with some optimization package.

If `probname.SIF` is correct, this results in the creation of the files

```
ELFUN.f, GROUP.f, RANGE.f, OUTSDIF.d, AUTOMAT.d
```

containing the decoded problem data. Possibly, a file called `EXTER.f` may also be created, if `probname.SIF` relies on external Fortran subroutines.

One of the less convenient features of SIF-encoded problems was that the decoding procedures in CUTE [BCGT95] were not designed to recognise, nor to alter, instance-dependent variable parameters such as problem dimensions or critical coefficients. Many real models, particularly those that arise from some form of discretization, depend upon parameters that a user might wish to refine. With CUTE, a user wishing to change such a parameter was forced to edit the SIF file—these files were usually provided with a number of suggested values, all but one of which were "commented out". Since a number of users found this to be very inconvenient, SifDec makes provisions both for the definition and for the altering of variable parameters from the problem-decoding scripts.

Any real or integer parameter definition containing the comment `$-PARAMETER` in field 5 (i.e., in columns 40-50) in a SIF file defines that parameter to be a *variable* parameter—this is consistent with old-style SIF-encoded problems since strings starting with `$` in this field were previously treated as comments. Any characters after `$-PARAMETER` will be regarded as comments, and will be passed back to a user on request. All SIF files in the CUTE collection that previously contained variable parameters have been updated to take advantage of this new SifDec facility, but of course they are still consistent with CUTE.

Given this extra syntax, the SIF decoding scripts have been extended to support two new options, allowing users to select variable parameters in the SIF file. The first of these options, `-show`, prints all the variable parameters present in the SIF file, along with suggested values to which they may be set as well as any other provided comments. For instance, the command

```
prompt% sifdecode -show LUBRIFC
```

produces, on my system, the output

```
  NN=10   (IE)                      comment:  n = 151 original value
  NN=50   (IE)                      comment:  n = 751
  NN=500  (IE)      -default value- comment:  n = 7501
```

indicating that the only parameter accepted by the problem described in `LUBRIFC.SIF` is called `NN`, that is takes integer values (IE), that it accepts the values 10, 50 or 500, 500 being the value used by default, and that these values correspond, in the problem, to the variable *n* being equal to 151, 751 or 7501 respectively. We now examine how to select one of the possible values for each such parameter.

The `-param` command-line option to `sifdecode` allows users to choose, from the command line, which values to assign to these parameters. For instance, assuming that `N` and `THETA` have been marked as variables parameters of `SAMPLE.SIF` and that `N=400` and `THETA=3.5` are valid values, the command

```
    prompt% sifdecode -param N=400,THETA=3.5 SAMPLE.SIF
```

will decode `SAMPLE.SIF` into the appropriate subroutines and data files, setting `N` to 400 and `THETA` to 3.5.
Note that the above command and

```
    prompt% sifdecode -param N=400 -param THETA=3.5 SAMPLE.SIF
```

are perfectly equivalent.

These new features allow users to systematically solve a set of problems in all prescribed, or possible,
sizes. Default values are given in each SIF file, and we have taken the opportunity to raise these defaults
to reflect the size of problem that we feel ought to be of current interest, given that many of the previous
defaults were assigned over ten years ago.

As a possible extension of the `-param` command-line option, users may force a problem to be de-
coded/solved using parameter values which have not necessarily been pre-assigned in the SIF file. This is
done using the `-force` option, as in

```
    prompt% sifdecode -param N=1000,THETA=3.5 -force SAMPLE.SIF
```

where `SAMPLE.SIF` does not contain the parameter setting N=1000. Omitting the `-force` option would
result in an abort of the process while specifying it results in the SIF decoder and the optimizer attempting
to complete the solve using the value 1000 for `N`. Note that nothing guarantees that this value is valid in
that context, and that the `-force` command-line option should be used carefully.

The options `-param` and `-force` are of course available on every sd*pak* interface and has the effect
described above. With consistency concerns in mind, the `-show` option is also available from every sd*pak*,
but has the sole effect of printing out the possible parameter settings, cancelling the subsequent call to the
package *pak*.

1.6 Attempting installation on an unsupported architecture

As far as UNIX-like platforms are concerned, it should not be too difficult to port SifDec. This might
require, however, a number of changes in several files. We suggest in this section where some of these
modifications could take place. Additional modifications may be necessary, depending on your local
system.

First, the installation scripts themselves may need to be altered, for compatibility reasons: the local C shell,
if there is one, may be different, or require different command-line options. For example, the very first
line of `install_sifdec` may be `#!/bin/csh` under Solaris, but has to be `#!/bin/csh -f` on LINUX
machines. All the scripts included in the SifDec distribution are thoroughly self-documented and should
be rather quickly understood by anyone familiar with the UNIX environment and the C shell. Similarly,
as all the SifDec scripts use the C shell, they may all need corresponding modifications.

Depending on your local architecture, you may have to create a new *machine*.cf file and alter `install_sifdec`
correspondingly. The *custom* size may itself be viewed to help design a new installation, as it may easily
suit your local hardware. The file where *custom* size may be tuned is `size.custom` and may be found in
`$SIFDEC/build/arch`.

Make sure that the last part of the file name

`compiler.`*machine*.*os*.*compiler*,

i.e. , the *compiler* part, is *identical* to the symbol representing this compiler in the Umake configuration files. Also make sure that new compilers that you define there appear in

`$CUTER/build/arch/f.arch` or `$CUTER/build/arch/c.arch`

for Fortran and CC++ compilers respectively, with matching symbols. More specifically, if you compiler name is *abc*, then the symbol which represents it in the configuration file must be "Isabc" and the block defining your compiler must look like

```
#ifdef Isabc
#define CompilerTagId          abc
#define umakeCompilerFlag       -DIsabc
#define CompileCmd             abc -c
#define LoadCmd               abc
#define CompilerIsF9095         yes
#define Compile9095Cmd         abc90 -c
#define Load9095Cmd           abc90
#define FortranFlags           -O
#define NumberOfBytes          8
#endif
```

If your system does not support man pages, these will be provided in pdf and other formats on the SifDec website, as will updates to this general documentation and other information.

Fortran 77 files should be standard and compatible for the most part. Check your local compiler documentation for possible incompatibilities. If there is no available Fortran 90 compiler on your platform, you will not be able to use those tools (unless you write one).

If your new installation procedure is a success, we will be pleased to include it in the next releases of SifDec, with proper credits. In this case, please send detailed information on your changes and on your local system. On the other hand, please feel free to contact us if you think we may be of some help.

Many thanks and again, good luck!

Chapter 2

Future versions of SifDec

2.1 Future features

- GUI,

- Have all the memory allocated in one place at the beginning. This would require further versions (like CUTEst) to be written in Fortran95, Fortran2000, or similar,

- SIF to AMPL converter,

- AMPL to SIF converter (maybe not),

- GAMS to SIF converter (even less likely),

- Support for Windows (double aaargh).

Chapter 3

License

Copyright (C) the Council for the Central Laboratory of the Research Councils, CERFACS and Facultes Universitaires Notre-Dame de la Paix (CCLRC, CERFACS and FUNDP) 2001.

SOFTWARE LICENSE AGREEMENT NOTICE - THIS SOFTWARE IS BEING PROVIDED TO YOU BY CERFACS UNDER THE FOLLOWING LICENSE. BY DOWN-LOADING, INSTALLING AND/OR USING THE SOFTWARE YOU AGREE THAT YOU HAVE READ, UNDERSTOOD AND WILL COMPLY WITH THESE FOLLOWING TERMS AND CONDITIONS.

1. This software program provided in source code format (the "Source Code") and any associated documentation (the "Documentation") are licensed, not sold, to you.

2. CCLRC, CERFACS and FUNDP grant you a personal, non-exclusive, non-transferable and royalty-free right to use, copy or modify the Source Code and Documentation, provided that you agree to comply with the terms and restrictions of this agreement. You may modify the Source Code and Documentation to make source code derivative works, object code derivative works and/or documentation derivative works (called "Derivative Works"). The Source Code, Documentation and Derivative Works (called "Licensed Software") may be used by you for personal and non-commercial use only. "non-commercial use" means uses that are not or will not result in the sale, lease or rental of the Licensed Software and/or the use of the Licensed Software in any commercial product or service. CCLRC, CERFACS and FUNDP reserve all rights not expressly granted to you. No other licenses are granted or implied.

3. The Source Code and Documentation are and will remain the sole property of CCLRC, CERFACS and FUNDP. The Source Code and Documentation are copyrighted works. You agree to treat any modification or derivative work of the Licensed Software as if it were part of the Licensed Software itself. In return for this license, you grant CCLRC, CERFACS and FUNDP a non-exclusive perpetual paid-up royalty-free license to make, sell, have made, copy, distribute and make derivative works of any modification or derivative work you make of the Licensed Software.

4. The licensee shall acknowledge the contribution of the Source Code in any publication of material dependent upon the use of the Source Code. The licensee shall use reasonable endeavours to send to CCLRC, CERFACS and FUNDP a copy of each such publication.
   For CCLRC, contact n.gould@rl.ac.uk, for CERFACS, contact orban@cerfacs.fr and for FUNDP, contact Philippe.Toint@fundp.ac.be.

23

5. CCLRC, CERFACS and FUNDP have no obligation to support the Licensed Software it is providing under this license.

   THE LICENSED SOFTWARE IS PROVIDED "AS IS" AND CCLRC, CERFACS and FUNDP MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CERFACS MAKE NO REPRESENTATIONS OR WAR-RANTIES OF MERCHANTIBILY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. CCLRC, CERFACS, FUNDP AND THE AUTHORS OF THE LICENSED SOFTWARE WILL NOT BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES, OR ANY OTHER RELIEF, OR FOR ANY CLAIM BY ANY THIRD PARTY, ARISING FROM YOUR USE OF THE LICENSED SOFTWARE.

6. This license is effective until terminated. You may terminate this license at any time by destroying the Licensed Software.

Appendix

Environment variables

The environment variables described in Table 3.1 are vital to SifDec. Refer to your local documentation or system administrator for more information on how to set these environment variables.

Name	Purpose
SIFDEC	Location of the source of the SifDec package;
MYSIFDEC	Location of the local instance of SifDec;
MASTSIF	Location of the local collection of SIF problems;

Table 3.1: Environment variables vital to SifDec.

Bibliography

[BCGT95] I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[CGT92] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. LANCELOT, *A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer-Verlag, 1992.

[Dub83] P. Dubois. *Software portability with imake*. O'Reilly & Associates, Inc., 1983.

[GOT01] N.I.M. Gould, D. Orban, and Ph.L. Toint. *General* CUTEr *documentation*. Rutherford Appleton Laboratory, UK, CERFACS, France and Facultés Universitaires Notre-Dame de la Paix, Belgium, 2001. see http://cuter.rl.ac.uk/cuter-www.