SUNDIALS Installation Guide

SUNDIALS v6.2.0

Eddy Banks[1], Alan C. Hindmarsh[1], Radu Serban[1], Cody J. Balos[1],
David J. Gardner[1], Daniel R. Reynolds[2], and Carol S. Woodward[1]

[1]*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*
[2]*Department of Mathematics, Southern Methodist University*

April 22, 2022

CONTRIBUTORS

The SUNDIALS library has been developed over many years by a number of contributors. The current SUNDIALS team consists of Cody J. Balos, David J. Gardner, Alan C. Hindmarsh, Daniel R. Reynolds, and Carol S. Woodward. We thank Radu Serban for significant and critical past contributions.

Other contributors to SUNDIALS include: James Almgren-Bell, Lawrence E. Banks, Peter N. Brown, George Byrne, Rujeko Chinomona, Scott D. Cohen, Aaron Collier, Keith E. Grant, Steven L. Lee, Shelby L. Lockhart, John Loffeld, Daniel McGreer, Slaven Peles, Cosmin Petra, H. Hunter Schwartz, Jean M. Sexton, Dan Shumaker, Steve G. Smith, Allan G. Taylor, Hilari C. Tiedeman, Chris White, Ting Yan, and Ulrike M. Yang.

Contents

Chapter 1

SUNDIALS Installation Procedure

The installation of any SUNDIALS package is accomplished by installing the SUNDIALS suite as a whole, according to the instructions that follow. The same procedure applies whether or not the downloaded file contains one or all solvers in SUNDIALS.

The SUNDIALS suite (or individual solvers) are distributed as compressed archives (`.tar.gz`). The name of the distribution archive is of the form `SOLVER-X.Y.Z.tar.gz`, where `SOLVER` is one of: `sundials`, `cvode`, `cvodes`, `arkode`, `ida`, `idas`, or `kinsol`, and `X.Y.Z` represents the version number (of the SUNDIALS suite or of the individual solver). To begin the installation, first uncompress and expand the sources, by issuing

```
% tar -zxf SOLVER-X.Y.Z.tar.gz
```

This will extract source files under a directory `SOLVER-X.Y.Z`.

Starting with version 2.6.0 of SUNDIALS, CMake is the only supported method of installation. The explanations of the installation procedure begin with a few common observations:

1. The remainder of this chapter will follow these conventions:

   `SOLVERDIR` is the directory `SOLVER-X.Y.Z` created above; i.e. the directory containing the SUNDIALS sources.

   `BUILDDIR` is the (temporary) directory under which SUNDIALS is built.

   `INSTDIR` is the directory under which the SUNDIALS exported header files and libraries will be installed. Typically, header files are exported under a directory `INSTDIR/include` while libraries are installed under `INSTDIR/lib`, with `INSTDIR` specified at configuration time.

2. For SUNDIALS' CMake-based installation, in-source builds are prohibited; in other words, the build directory `BUILDDIR` can **not** be the same as `SOLVERDIR` and such an attempt will lead to an error. This prevents "polluting" the source tree and allows efficient builds for different configurations and/or options.

3. The installation directory `INSTDIR` can not be the same as the source directory `SOLVERDIR`.

4. By default, only the libraries and header files are exported to the installation directory `INSTDIR`. If enabled by the user (with the appropriate toggle for CMake), the examples distributed with SUNDIALS will be built together with the solver libraries but the installation step will result in exporting (by default in a subdirectory of the installation directory) the example sources and sample outputs together with automatically generated configuration files that reference the *installed* SUNDIALS headers and libraries. As such, these configuration files for the SUNDIALS examples can be used as "templates" for your own problems. CMake installs `CMakeLists.txt` files and also (as an option available only under Unix/Linux) `Makefile` files. Note this installation approach also allows the option of building the SUNDIALS examples without having to install them. (This can be used as a sanity check for the freshly built libraries.)

Further details on the CMake-based installation procedures, instructions for manual compilation, and a roadmap of the resulting installed libraries and exported header files, are provided in §1.1 and §1.2.

1.1 CMake-based installation

CMake-based installation provides a platform-independent build system. CMake can generate Unix and Linux Makefiles, as well as KDevelop, Visual Studio, and (Apple) XCode project files from the same configuration file. In addition, CMake also provides a GUI front end and which allows an interactive build and installation process.

The SUNDIALS build process requires CMake version 3.12.0 or higher and a working C compiler. On Unix-like operating systems, it also requires Make (and `curses`, including its development libraries, for the GUI front end to CMake, `ccmake` or `cmake-gui`), while on Windows it requires Visual Studio. While many Linux distributions offer CMake, the version included may be out of date. CMake adds new features regularly, and you should download the latest version from http://www.cmake.org. Build instructions for CMake (only necessary for Unix-like systems) can be found on the CMake website. Once CMake is installed, Linux/Unix users will be able to use `ccmake` or `cmake-gui` (depending on the version of CMake), while Windows users will be able to use `CMakeSetup`.

As previously noted, when using CMake to configure, build and install SUNDIALS, it is always required to use a separate build directory. While in-source builds are possible, they are explicitly prohibited by the SUNDIALS CMake scripts (one of the reasons being that, unlike autotools, CMake does not provide a `make distclean` procedure and it is therefore difficult to clean-up the source tree after an in-source build). By ensuring a separate build directory, it is an easy task for the user to clean-up all traces of the build by simply removing the build directory. CMake does generate a `make clean` which will remove files generated by the compiler and linker.

1.1.1 Configuring, building, and installing on Unix-like systems

The default CMake configuration will build all included solvers and associated examples and will build static and shared libraries. The INSTDIR defaults to `/usr/local` and can be changed by setting the `CMAKE_INSTALL_PREFIX` variable. Support for FORTRAN and all other options are disabled.

CMake can be used from the command line with the `cmake` command, or from a `curses`-based GUI by using the `ccmake` command, or from a wxWidgets or QT based GUI by using the `cmake-gui` command. Examples for using both text and graphical methods will be presented. For the examples shown it is assumed that there is a top level SUNDIALS directory with appropriate source, build and install directories:

```
$ mkdir (...)/INSTDIR
$ mkdir (...)/BUILDDIR
$ cd (...)/BUILDDIR
```

1.1.1.1 Building with the GUI

Using CMake with the `ccmake` GUI follows the general process:

1. Select and modify values, run configure (`c` key)

2. New values are denoted with an asterisk

3. To set a variable, move the cursor to the variable and press enter

   - If it is a boolean (ON/OFF) it will toggle the value

   - If it is string or file, it will allow editing of the string

   - For file and directories, the `<tab>` key can be used to complete

4. Repeat until all values are set as desired and the generate option is available (`g` key)

5. Some variables (advanced variables) are not visible right away; to see advanced variables, toggle to advanced mode (`t` key)

6. To search for a variable press the `/` key, and to repeat the search, press the `n` key

Using CMake with the `cmake-gui` GUI follows a similar process:

1. Select and modify values, click `Configure`

2. The first time you click `Configure`, make sure to pick the appropriate generator (the following will assume generation of Unix Makfiles).

3. New values are highlighted in red

4. To set a variable, click on or move the cursor to the variable and press enter

   - If it is a boolean (`ON/OFF`) it will check/uncheck the box

   - If it is string or file, it will allow editing of the string. Additionally, an ellipsis button will appear `...` on the far right of the entry. Clicking this button will bring up the file or directory selection dialog.

   - For files and directories, the `<tab>` key can be used to complete

5. Repeat until all values are set as desired and click the `Generate` button

6. Some variables (advanced variables) are not visible right away; to see advanced variables, click the `advanced` button

To build the default configuration using the curses GUI, from the `BUILDDIR` enter the `ccmake` command and point to the `SOLVERDIR`:

```
$ ccmake (...)/SOLVERDIR
```

Similarly, to build the default configuration using the wxWidgets GUI, from the `BUILDDIR` enter the `cmake-gui` command and point to the `SOLVERDIR`:

```
$ cmake-gui (...)/SOLVERDIR
```

The default curses configuration screen is shown in the following figure.

The default INSTDIR for both SUNDIALS and the corresponding examples can be changed by setting the `CMAKE_-INSTALL_PREFIX` and the `EXAMPLES_INSTALL_PATH` as shown in the following figure.

Pressing the `g` key or clicking `generate` will generate Makefiles including all dependencies and all rules to build SUNDIALS on this system. Back at the command prompt, you can now run:

```
$ make
```

or for a faster parallel build (e.g. using 4 threads), you can run

```
$ make -j 4
```

To install SUNDIALS in the installation directory specified in the configuration, simply run:

```
$ make install
```

```
                                        Page 1 of 1
BUILD_ARKODE                          *ON
BUILD_CVODE                           *ON
BUILD_CVODES                          *ON
BUILD_EXAMPLES                        *ON
BUILD_IDA                             *ON
BUILD_IDAS                            *ON
BUILD_KINSOL                          *ON
BUILD_SHARED_LIBS                     *ON
BUILD_STATIC_LIBS                     *ON
BUILD_TESTING                         *ON
CMAKE_BUILD_TYPE                      *
CMAKE_CXX_COMPILER                    */usr/bin/c++
CMAKE_CXX_FLAGS                       *
CMAKE_C_COMPILER                      */usr/bin/cc
CMAKE_C_FLAGS                         *
CMAKE_INSTALL_LIBDIR                  *lib64
CMAKE_INSTALL_PREFIX                  */usr/local
ENABLE_CUDA                           *OFF
ENABLE_FORTRAN                        *OFF
ENABLE_HYPRE                          *OFF
ENABLE_KLU                            *OFF
ENABLE_LAPACK                         *OFF
ENABLE_MPI                            *OFF
ENABLE_OPENMP                         *OFF
ENABLE_OPENMP_DEVICE                  *OFF
ENABLE_PETSC                          *OFF
ENABLE_PTHREAD                        *OFF
ENABLE_RAJA                           *OFF
ENABLE_SUPERLUDIST                    *OFF
ENABLE_SUPERLUMT                      *OFF
ENABLE_TRILINOS                       *OFF
EXAMPLES_ENABLE_C                     *ON
EXAMPLES_ENABLE_CXX                   *ON
EXAMPLES_INSTALL                      *ON
EXAMPLES_INSTALL_PATH                 */usr/local/examples
SUNDIALS_BUILD_WITH_MONITORING        *OFF
SUNDIALS_INDEX_SIZE                   *64
SUNDIALS_PRECISION                    *DOUBLE
USE_GENERIC_MATH                      *ON
USE_XSDK_DEFAULTS                     *OFF



BUILD_ARKODE: Build the ARKODE library
Press [enter] to edit option Press [d] to delete an entry                    CMake Version 3.12.1
Press [c] to configure
Press [h] for help         Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

Fig. 1.1: Default configuration screen. Note: Initial screen is empty. To get this default configuration, press 'c' repeatedly (accepting default values denoted with asterisk) until the 'g' option is available.

```
                                       Page 1 of 1
BUILD_ARKODE                        *ON
BUILD_CVODE                         *ON
BUILD_CVODES                        *ON
BUILD_EXAMPLES                      *ON
BUILD_IDA                           *ON
BUILD_IDAS                          *ON
BUILD_KINSOL                        *ON
BUILD_SHARED_LIBS                   *ON
BUILD_STATIC_LIBS                   *ON
BUILD_TESTING                       *ON
CMAKE_BUILD_TYPE                    *
CMAKE_CXX_COMPILER                  */usr/bin/c++
CMAKE_CXX_FLAGS                     *
CMAKE_C_COMPILER                    */usr/bin/cc
CMAKE_C_FLAGS                       *
CMAKE_INSTALL_LIBDIR                *lib64
CMAKE_INSTALL_PREFIX                */usr/casc/sundials/instdir
ENABLE_CUDA                         *OFF
ENABLE_FORTRAN                      *OFF
ENABLE_HYPRE                        *OFF
ENABLE_KLU                          *OFF
ENABLE_LAPACK                       *OFF
ENABLE_MPI                          *OFF
ENABLE_OPENMP                       *OFF
ENABLE_OPENMP_DEVICE                *OFF
ENABLE_PETSC                        *OFF
ENABLE_PTHREAD                      *OFF
ENABLE_RAJA                         *OFF
ENABLE_SUPERLUDIST                  *OFF
ENABLE_SUPERLUMT                    *OFF
ENABLE_TRILINOS                     *OFF
EXAMPLES_ENABLE_C                   *ON
EXAMPLES_ENABLE_CXX                 *ON
EXAMPLES_INSTALL                    *ON
EXAMPLES_INSTALL_PATH               */usr/casc/sundials/instdir/examples
SUNDIALS_BUILD_WITH_MONITORING      *OFF
SUNDIALS_INDEX_SIZE                 *64
SUNDIALS_PRECISION                  *DOUBLE
USE_GENERIC_MATH                    *ON
USE_XSDK_DEFAULTS                   *OFF



EXAMPLES_INSTALL_PATH: Output directory for installing example files
Press [enter] to edit option Press [d] to delete an entry                    CMake Version 3.12.1
Press [c] to configure
Press [h] for help          Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

Fig. 1.2: Changing the INSTDIR for SUNDIALS and corresponding EXAMPLES.

1.1.1.2 Building from the command line

Using CMake from the command line is simply a matter of specifying CMake variable settings with the `cmake` command. The following will build the default configuration:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/home/myname/sundials/instdir \
>  -DEXAMPLES_INSTALL_PATH=/home/myname/sundials/instdir/examples \
>  ../srcdir
$ make
$ make install
```

1.1.2 Configuration options (Unix/Linux)

A complete list of all available options for a CMake-based SUNDIALS configuration is provide below. Note that the default values shown are for a typical configuration on a Linux system and are provided as illustration only.

BUILD_ARKODE
> Build the ARKODE library
>
> Default: `ON`

BUILD_CVODE
> Build the CVODE library
>
> Default: `ON`

BUILD_CVODES
> Build the CVODES library
>
> Default: `ON`

BUILD_IDA
> Build the IDA library
>
> Default: `ON`

BUILD_IDAS
> Build the IDAS library
>
> Default: `ON`

BUILD_KINSOL
> Build the KINSOL library
>
> Default: `ON`

BUILD_SHARED_LIBS
> Build shared libraries
>
> Default: `ON`

BUILD_STATIC_LIBS
> Build static libraries
>
> Default: `ON`

CMAKE_BUILD_TYPE
> Choose the type of build, options are: `None`, `Debug`, `Release`, `RelWithDebInfo`, and `MinSizeRel`
>
> Default:

> **Note:** Specifying a build type will trigger the corresponding build type specific compiler flag options below which will be appended to the flags set by `CMAKE_<language>_FLAGS`.

CMAKE_C_COMPILER
> C compiler
>
> Default: `/usr/bin/cc`

CMAKE_C_FLAGS
> Flags for C compiler
>
> Default:

CMAKE_C_FLAGS_DEBUG
> Flags used by the C compiler during debug builds
>
> Default: `-g`

CMAKE_C_FLAGS_MINSIZEREL
> Flags used by the C compiler during release minsize builds
>
> Default: `-Os -DNDEBUG`

CMAKE_C_FLAGS_RELEASE
> Flags used by the C compiler during release builds
>
> Default: `-O3 -DNDEBUG`

CMAKE_C_STANDARD
> The C standard to build C parts of SUNDIALS with.
>
> Default: 99
>
> Options: 90, 99, 11, 17.

CMAKE_C_EXTENSIONS
> Enable compiler specific C extensions.
>
> Default: `OFF`

CMAKE_CXX_COMPILER
> C++ compiler
>
> Default: `/usr/bin/c++`

> **Note:** A C++ compiler is only required when a feature requiring C++ is enabled (e.g., CUDA, HIP, SYCL, RAJA, etc.) or the C++ examples are enabled.
>
> All SUNDIALS solvers can be used from C++ applications without setting any additional configuration options.

CMAKE_CXX_FLAGS
> Flags for C++ compiler
>
> Default:

CMAKE_CXX_FLAGS_DEBUG
> Flags used by the C++ compiler during debug builds
>
> Default: `-g`

CMAKE_CXX_FLAGS_MINSIZEREL

    Flags used by the C++ compiler during release minsize builds

    Default: `-Os -DNDEBUG`

CMAKE_CXX_FLAGS_RELEASE

    Flags used by the C++ compiler during release builds

    Default: `-O3 -DNDEBUG`

CMAKE_CXX_STANDARD

    The C++ standard to build C++ parts of SUNDIALS with.

    Default: 11

    Options: 98, 11, 14, 17, 20.

CMAKE_CXX_EXTENSIONS

    Enable compiler specific C++ extensions.

    Default: `OFF`

CMAKE_Fortran_COMPILER

    Fortran compiler

    Default: `/usr/bin/gfortran`

Note: Fortran support (and all related options) are triggered only if either Fortran-C support (`BUILD_FORTRAN_-MODULE_INTERFACE`) or LAPACK (`ENABLE_LAPACK`) support is enabled.

CMAKE_Fortran_FLAGS

    Flags for Fortran compiler

    Default:

CMAKE_Fortran_FLAGS_DEBUG

    Flags used by the Fortran compiler during debug builds

    Default: `-g`

CMAKE_Fortran_FLAGS_MINSIZEREL

    Flags used by the Fortran compiler during release minsize builds

    Default: `-Os`

CMAKE_Fortran_FLAGS_RELEASE

    Flags used by the Fortran compiler during release builds

    Default: `-O3`

CMAKE_INSTALL_LIBDIR

    The directory under which libraries will be installed.

    Default: Set based on the system: `lib`, `lib64`, or `lib/<multiarch-tuple>`

CMAKE_INSTALL_PREFIX

    Install path prefix, prepended onto install directories

    Default: `/usr/local`

Note: The user must have write access to the location specified through this option. Exported SUNDIALS header files and libraries will be installed under subdirectories `include` and `lib` of `CMAKE_INSTALL_PREFIX`,

---

respectively.

ENABLE_CUDA
> Build the SUNDIALS CUDA modules.
>
> Default: `OFF`

CMAKE_CUDA_ARCHITECTURES
> Specifies the CUDA architecture to compile for.
>
> Default: `sm_30`

ENABLE_XBRAID
> Enable or disable the ARKStep + XBraid interface.
>
> Default: `OFF`

> **Note:** See additional information on building with *XBraid* enabled in §1.1.4.

EXAMPLES_ENABLE_C
> Build the SUNDIALS C examples
>
> Default: `ON`

EXAMPLES_ENABLE_CXX
> Build the SUNDIALS C++ examples
>
> Default: `OFF`

EXAMPLES_ENABLE_CUDA
> Build the SUNDIALS CUDA examples
>
> Default: `OFF`

> **Note:** You need to enable CUDA support to build these examples.

EXAMPLES_ENABLE_F2003
> Build the SUNDIALS Fortran2003 examples
>
> Default: `ON` (if `BUILD_FORTRAN_MODULE_INTERFACE` is `ON`)

EXAMPLES_INSTALL
> Install example files
>
> Default: `ON`

> **Note:** This option is triggered when any of the SUNDIALS example programs are enabled (`EXAMPLES_-ENABLE_<language>` is `ON`). If the user requires installation of example programs then the sources and sample output files for all SUNDIALS modules that are currently enabled will be exported to the directory specified by `EXAMPLES_INSTALL_PATH`. A CMake configuration script will also be automatically generated and exported to the same directory. Additionally, if the configuration is done under a Unix-like system, makefiles for the compilation of the example programs (using the installed SUNDIALS libraries) will be automatically generated and exported to the directory specified by `EXAMPLES_INSTALL_PATH`.

EXAMPLES_INSTALL_PATH
> Output directory for installing example files

Default: `/usr/local/examples`

Note: The actual default value for this option will be an `examples` subdirectory created under `CMAKE_IN-STALL_PREFIX`.

BUILD_FORTRAN_MODULE_INTERFACE
  Enable Fortran2003 interface

  Default: `OFF`

ENABLE_HYPRE
  Flag to enable *hypre* support

  Default: `OFF`

Note: See additional information on building with *hypre* enabled in §1.1.4.

HYPRE_INCLUDE_DIR
  Path to *hypre* header files

  Default: none

HYPRE_LIBRARY
  Path to *hypre* installed library files

  Default: none

ENABLE_KLU
  Enable KLU support

  Default: `OFF`

Note: See additional information on building with KLU enabled in §1.1.4.

KLU_INCLUDE_DIR
  Path to SuiteSparse header files

  Default: none

KLU_LIBRARY_DIR
  Path to SuiteSparse installed library files

  Default: none

ENABLE_LAPACK
  Enable LAPACK support

  Default: `OFF`

Note: Setting this option to `ON` will trigger additional CMake options. See additional information on building with LAPACK enabled in §1.1.4.

LAPACK_LIBRARIES
  LAPACK (and BLAS) libraries

  Default: `/usr/lib/liblapack.so;/usr/lib/libblas.so`

Note: CMake will search for libraries in your `LD_LIBRARY_PATH` prior to searching default system paths.

`ENABLE_MAGMA`

Enable MAGMA support.

Default: `OFF`

Note: Setting this option to `ON` will trigger additional options related to MAGMA.

`MAGMA_DIR`

Path to the root of a MAGMA installation.

Default: none

`SUNDIALS_MAGMA_BACKENDS`

Which MAGMA backend to use under the SUNDIALS MAGMA interface.

Default: `CUDA`

`ENABLE_MPI`

Enable MPI support. This will build the parallel nvector and the MPI-aware version of the ManyVector library.

Default: `OFF`

Note: Setting this option to `ON` will trigger several additional options related to MPI.

`MPI_C_COMPILER`

`mpicc` program

Default:

`MPI_CXX_COMPILER`

`mpicxx` program

Default:

Note: This option is triggered only if MPI is enabled (`ENABLE_MPI` is `ON`) and C++ examples are enabled (`EXAMPLES_ENABLE_CXX` is `ON`). All SUNDIALS solvers can be used from C++ MPI applications by default without setting any additional configuration options other than `ENABLE_MPI`.

`MPI_Fortran_COMPILER`

`mpif90` program

Default:

Note: This option is triggered only if MPI is enabled (`ENABLE_MPI` is `ON`) and Fortran-C support is enabled (`EXAMPLES_ENABLE_F2003` is `ON`).

`MPIEXEC_EXECUTABLE`

Specify the executable for running MPI programs

Default: `mpirun`

> **Note:** This option is triggered only if MPI is enabled (`ENABLE_MPI` is `ON`).

ENABLE_ONEMKL
  Enable oneMKL support.

  Default: `OFF`

ONEMKL_DIR
  Path to oneMKL installation.

  Default: none

ENABLE_OPENMP
  Enable OpenMP support (build the OpenMP NVector)

  Default: `OFF`

ENABLE_PETSC
  Enable PETSc support

  Default: `OFF`

> **Note:** See additional information on building with PETSc enabled in §1.1.4.

PETSC_DIR
  Path to PETSc installation

  Default: none

PETSC_LIBRARIES
  Semi-colon separated list of PETSc link libraries. Unless provided by the user, this is autopopulated based on the PETSc installation found in `PETSC_DIR`.

  Default: none

PETSC_INCLUDES
  Semi-colon separated list of PETSc include directroies. Unless provided by the user, this is autopopulated based on the PETSc installation found in `PETSC_DIR`.

  Default: none

ENABLE_PTHREAD
  Enable Pthreads support (build the Pthreads NVector)

  Default: `OFF`

ENABLE_RAJA
  Enable RAJA support.

  Default: OFF

> **Note:** You need to enable CUDA or HIP in order to build the RAJA vector module.

SUNDIALS_RAJA_BACKENDS
  If building SUNDIALS with RAJA support, this sets the RAJA backend to target. Values supported are CUDA, HIP, or SYCL.

  Default: CUDA

ENABLE_SUPERLUDIST

Enable SuperLU_DIST support

Default: `OFF`

Note: See additional information on building wtih SuperLU_DIST enabled in §1.1.4.

SUPERLUDIST_INCLUDE_DIR

Path to SuperLU_DIST header files (under a typical SuperLU_DIST install, this is typically the SuperLU_DIST SRC directory)

Default: none

SUPERLUDIST_LIBRARY_DIR

Path to SuperLU_DIST installed library files

Default: none

SUPERLUDIST_LIBRARIES

Semi-colon separated list of libraries needed for SuperLU_DIST

Default: none

SUPERLUDIST_OpenMP

Enable SUNDIALS support for SuperLU_DIST built with OpenMP

Default: none

Note: SuperLU_DIST must be built with OpenMP support for this option to function. Additionally the environment variable `OMP_NUM_THREADS` must be set to the desired number of threads.

ENABLE_SUPERLUMT

Enable SuperLU_MT support

Default: `OFF`

Note: See additional information on building with SuperLU_MT enabled in §1.1.4.

SUPERLUMT_INCLUDE_DIR

Path to SuperLU_MT header files (under a typical SuperLU_MT install, this is typically the SuperLU_MT SRC directory)

Default: none

SUPERLUMT_LIBRARY_DIR

Path to SuperLU_MT installed library files

Default: none

SUPERLUMT_THREAD_TYPE

Must be set to Pthread or OpenMP, depending on how SuperLU_MT was compiled.

Default: Pthread

ENABLE_SYCL

Enable SYCL support.

Default: OFF

Note: At present the only supported SYCL compiler is the DPC++ (Intel oneAPI) compiler. CMake does not currently support autodetection of SYCL compilers and `CMAKE_CXX_COMPILER` must be set to a valid SYCL compiler i.e., `dpcpp` in order to build with SYCL support.

SUNDIALS_LOGGING_LEVEL

Set the maximum logging level for the SUNLogger runtime API. The higher this is set, the more output that may be logged, and the more performance may degrade. The options are:

- `0` – no logging

- `1` – log errors

- `2` – log errors + warnings

- `3` – log errors + warnings + informational output

- `4` – log errors + warnings + informational output + debug output

- `5` – log all of the above and even more (e.g. vector valued variables may be logged)

Default: 0

SUNDIALS_LOGGING_ENABLE_MPI

Enables MPI support in the SUNLogger runtime API. I.e., makes the logger MPI aware and capable of outputting only on specific ranks.

Default: *ENABLE_MPI*

Note: The logger may be used in an MPI application without MPI support turned on, but it will output on all ranks.

SUNDIALS_BUILD_WITH_MONITORING

Build SUNDIALS with capabilties for fine-grained monitoring of solver progress and statistics. This is primarily useful for debugging.

Default: OFF

Warning: Building with monitoring may result in minor performance degradation even if monitoring is not utilized.

SUNDIALS_BUILD_WITH_PROFILING

Build SUNDIALS with capabilties for fine-grained profiling.

Default: OFF

Warning: Profiling will impact performance, and should be enabled judiciously.

ENABLE_CALIPER

Enable CALIPER support

Default: OFF

Note: Using Caliper requires setting *SUNDIALS_BUILD_WITH_PROFILING* to `ON`.

CALIPER_DIR
> Path to the root of a Caliper installation
>
> Default: None

SUNDIALS_F77_FUNC_CASE
> Specify the case to use in the Fortran name-mangling scheme, options are: `lower` or `upper`
>
> Default:
>
> ---
> **Note:** The build system will attempt to infer the Fortran name-mangling scheme using the Fortran compiler. This option should only be used if a Fortran compiler is not available or to override the inferred or default (`lower`) scheme if one can not be determined. If used, `SUNDIALS_F77_FUNC_UNDERSCORES` must also be set.
> ---

SUNDIALS_F77_FUNC_UNDERSCORES
> Specify the number of underscores to append in the Fortran name-mangling scheme, options are: `none`, `one`, or `two`
>
> Default:
>
> ---
> **Note:** The build system will attempt to infer the Fortran name-mangling scheme using the Fortran compiler. This option should only be used if a Fortran compiler is not available or to override the inferred or default (`one`) scheme if one can not be determined. If used, `SUNDIALS_F77_FUNC_CASE` must also be set.
> ---

SUNDIALS_INDEX_TYPE
> Integer type used for SUNDIALS indices. The size must match the size provided for the `SUNDIALS_INDEX_SIZE` option.
>
> Default: Automatically determined based on *SUNDIALS_INDEX_SIZE*
>
> ---
> **Note:** In past SUNDIALS versions, a user could set this option to `INT64_T` to use 64-bit integers, or `INT32_T` to use 32-bit integers. Starting in SUNDIALS 3.2.0, these special values are deprecated. For SUNDIALS 3.2.0 and up, a user will only need to use the *SUNDIALS_INDEX_SIZE* option in most cases.
> ---

SUNDIALS_INDEX_SIZE
> Integer size (in bits) used for indices in SUNDIALS, options are: `32` or `64`
>
> Default: `64`
>
> ---
> **Note:** The build system tries to find an integer type of appropriate size. Candidate 64-bit integer types are (in order of preference): `int64_t`, `__int64`, `long long`, and `long`. Candidate 32-bit integers are (in order of preference): `int32_t`, `int`, and `long`. The advanced option, *SUNDIALS_INDEX_TYPE* can be used to provide a type not listed here.
> ---

SUNDIALS_PRECISION
> The floating-point precision used in SUNDIALS packages and class implementations, options are: `double`, `single`, or `extended`
>
> Default: `double`

SUNDIALS_INSTALL_CMAKEDIR
> Installation directory for the SUNDIALS cmake files (relative to *CMAKE_INSTALL_PREFIX*).
>
> Default: `CMAKE_INSTALL_PREFIX/cmake/sundials`

USE_GENERIC_MATH

      Use generic (`stdc`) math libraries

      Default: `ON`

XBRAID_DIR

      The root directory of the XBraid installation.

      Default: `OFF`

XBRAID_INCLUDES

      Semi-colon separated list of XBraid include directories. Unless provided by the user, this is autopopulated based on the XBraid installation found in `XBRAID_DIR`.

      Default: none

XBRAID_LIBRARIES

      Semi-colon separated list of XBraid link libraries. Unless provided by the user, this is autopopulated based on the XBraid installation found in `XBRAID_DIR`.

      Default: none

USE_XSDK_DEFAULTS

      Enable xSDK (see https://xsdk.info for more information) default configuration settings. This sets `CMAKE_-BUILD_TYPE` to `Debug`, `SUNDIALS_INDEX_SIZE` to 32 and `SUNDIALS_PRECISION` to double.

      Default: `OFF`

1.1.3 Configuration examples

The following examples will help demonstrate usage of the CMake configure options.

To configure SUNDIALS using the default C and Fortran compilers, and default `mpicc` and `mpif90` parallel compilers, enable compilation of examples, and install libraries, headers, and example sources under subdirectories of `/home/myname/sundials/`, use:

```
% cmake \
> -DCMAKE_INSTALL_PREFIX=/home/myname/sundials/instdir \
> -DEXAMPLES_INSTALL_PATH=/home/myname/sundials/instdir/examples \
> -DENABLE_MPI=ON \
> /home/myname/sundials/srcdir

% make install
```

To disable installation of the examples, use:

```
% cmake \
> -DCMAKE_INSTALL_PREFIX=/home/myname/sundials/instdir \
> -DEXAMPLES_INSTALL_PATH=/home/myname/sundials/instdir/examples \
> -DENABLE_MPI=ON \
> -DEXAMPLES_INSTALL=OFF \
> /home/myname/sundials/srcdir

% make install
```

1.1.4 Working with external Libraries

The SUNDIALS suite contains many options to enable implementation flexibility when developing solutions. The following are some notes addressing specific configurations when using the supported third party libraries.

1.1.4.1 Building with LAPACK

To enable LAPACK, set the `ENABLE_LAPACK` option to `ON`. If the directory containing the LAPACK library is in the `LD_LIBRARY_PATH` environment variable, CMake will set the `LAPACK_LIBRARIES` variable accordingly, otherwise CMake will attempt to find the LAPACK library in standard system locations. To explicitly tell CMake what library to use, the `LAPACK_LIBRARIES` variable can be set to the desired libraries required for LAPACK.

```
% cmake \
> -DCMAKE_INSTALL_PREFIX=/home/myname/sundials/instdir \
> -DEXAMPLES_INSTALL_PATH=/home/myname/sundials/instdir/examples \
> -DENABLE_LAPACK=ON \
> -DLAPACK_LIBRARIES=/mylapackpath/lib/libblas.so;/mylapackpath/lib/liblapack.so \
> /home/myname/sundials/srcdir

% make install
```

Note: If a working Fortran compiler is not available to infer the Fortran name-mangling scheme, the options `SUNDI-ALS_F77_FUNC_CASE` and `SUNDIALS_F77_FUNC_UNDERSCORES` *must* be set in order to bypass the check for a Fortran compiler and define the name-mangling scheme. The defaults for these options in earlier versions of SUNDIALS were `lower` and `one`, respectively.

SUNDIALS has been tested with OpenBLAS 0.3.18.

1.1.4.2 Building with KLU

KLU is a software package for the direct solution of sparse nonsymmetric linear systems of equations that arise in circuit simulation and is part of SuiteSparse, a suite of sparse matrix software. The library is developed by Texas A&M University and is available from the SuiteSparse GitHub repository.

To enable KLU, set `ENABLE_KLU` to `ON`, set `KLU_INCLUDE_DIR` to the `include` path of the KLU installation and set `KLU_LIBRARY_DIR` to the `lib` path of the KLU installation. The CMake configure will result in populating the following variables: `AMD_LIBRARY`, `AMD_LIBRARY_DIR`, `BTF_LIBRARY`, `BTF_LIBRARY_DIR`, `COLAMD_LIBRARY`, `CO-LAMD_LIBRARY_DIR`, and `KLU_LIBRARY`.

SUNDIALS has been tested with SuiteSparse version 5.10.1.

1.1.4.3 Building with SuperLU_DIST

SuperLU_DIST is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations in a distributed memory setting. The library is developed by Lawrence Berkeley National Laboratory and is available from the SuperLU_DIST GitHub repository.

To enable SuperLU_DIST, set `ENABLE_SUPERLUDIST` to `ON`, set `SUPERLUDIST_INCLUDE_DIR` to the SRC path of the SuperLU_DIST installation, and set the variable `SUPERLUMT_LIBRARY_DIR` to the `lib` path of the SuperLU_DIST installation. At the same time, the variable `SUPERLUDIST_LIBRARIES` must be set to a semi-colon separated list of other libraries SuperLU_DIST depends on. For example, if SuperLU_DIST was built with LAPACK, then include the

---

LAPACK library in this list. If SuperLU_DIST was built with OpenMP support, then you may set `SUPERLUDIST_-OpenMP` to `ON` utilize the OpenMP functionality of SuperLU_DIST.

SUNDIALS has been tested with SuperLU_DIST 7.1.1.

1.1.4.4 Building with SuperLU_MT

SuperLU_MT is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on shared memory parallel machines. The library is developed by Lawrence Berkeley National Laboratory and is available from the SuperLU_MT GitHub repository.

To enable SuperLU_MT, set `ENABLE_SUPERLUMT` to `ON`, set `SUPERLUMT_INCLUDE_DIR` to the SRC path of the SuperLU_MT installation, and set the variable `SUPERLUMT_LIBRARY_DIR` to the `lib` path of the SuperLU_MT installation. At the same time, the variable `SUPERLUMT_LIBRARIES` must be set to a semi-colon separated list of other libraries SuperLU_MT depends on. For example, if SuperLU_MT was build with an external blas library, then include the full path to the blas library in this list. Additionally, the variable `SUPERLUMT_THREAD_TYPE` must be set to either `Pthread` or `OpenMP`.

Do not mix thread types when building SUNDIALS solvers. If threading is enabled for SUNDIALS by having either `ENABLE_OPENMP` or `ENABLE_PTHREAD` set to `ON` then SuperLU_MT should be set to use the same threading type.

SUNDIALS has been tested with SuperLU_MT version 3.1.

1.1.4.5 Building with PETSc

The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines for simulating applications modeled by partial differential equations. The library is developed by Argonne National Laboratory and is available from the PETSc GitLab repository.

To enable PETSc, set `ENABLE_PETSC` to `ON`, and set `PETSC_DIR` to the path of the PETSc installation. Alternatively, a user can provide a list of include paths in `PETSC_INCLUDES` and a list of complete paths to the PETSc libraries in `PETSC_LIBRARIES`.

SUNDIALS has been tested with PETSc version 3.16.1.

1.1.4.6 Building with *hypre*

hypre is a library of high performance preconditioners and solvers featuring multigrid methods for the solution of large, sparse linear systems of equations on massively parallel computers. The library is developed by Lawrence Livermore National Laboratory and is available from the hypre GitHub repository.

To enable *hypre*, set `ENABLE_HYPRE` to `ON`, set `HYPRE_INCLUDE_DIR` to the `include` path of the *hypre* installation, and set the variable `HYPRE_LIBRARY_DIR` to the `lib` path of the *hypre* installation.

Note: SUNDIALS must be configured so that `SUNDIALS_INDEX_SIZE` is compatible with `HYPRE_BigInt` in the *hypre* installation.

SUNDIALS has been tested with *hypre* version 2.23.0

1.1.4.7 Building with MAGMA

The Matrix Algebra on GPU and Multicore Architectures (MAGMA) project provides a dense linear algebra library similar to LAPACK but targeting heterogeneous architectures. The library is developed by the University of Tennessee and is available from the UTK webpage.

To enable the SUNDIALS MAGMA interface set `ENABLE_MAGMA` to `ON`, `MAGMA_DIR` to the MAGMA installation path, and `SUNDIALS_MAGMA_BACKENDS` to the desired MAGMA backend to use with SUNDIALS e.g., `CUDA` or `HIP`.

SUNDIALS has been tested with MAGMA version 2.6.1.

1.1.4.8 Building with oneMKL

The Intel oneAPI Math Kernel Library (oneMKL) includes CPU and DPC++ interfaces for LAPACK dense linear algebra routines. The SUNDIALS oneMKL interface targets the DPC++ routines, to utilize the CPU routine see §1.1.4.1.

To enable the SUNDIALS oneMKL interface set `ENABLE_ONEMKL` to `ON` and `ONEMKL_DIR` to the oneMKL installation path.

SUNDIALS has been tested with oneMKL version 2021.4.

1.1.4.9 Building with CUDA

The NVIDIA CUDA Toolkit provides a development environment for GPU-accelerated computing with NVIDIA GPUs. The CUDA Toolkit and compatible NVIDIA drivers are available from the NVIDIA developer website.

To enable CUDA, set `ENABLE_CUDA` to `ON`. If CUDA is installed in a nonstandard location, you may be prompted to set the variable `CUDA_TOOLKIT_ROOT_DIR` with your CUDA Toolkit installation path. To enable CUDA examples, set `EXAMPLES_ENABLE_CUDA` to `ON`.

SUNDIALS has been tested with the CUDA toolkit versions 10 and 11.

1.1.4.10 Building with RAJA

RAJA is a performance portability layer developed by Lawrence Livermore National Laboratory and can be obtained from the RAJA GitHub repository.

Building SUNDIALS RAJA modules requires a CUDA, HIP, or SYCL enabled RAJA installation. To enable RAJA, set `ENABLE_RAJA` to `ON`, set `SUNDIALS_RAJA_BACKENDS` to the desired backend (`CUDA`, `HIP`, or `SYCL`), and set `ENABLE_-CUDA`, `ENABLE_HIP`, or `ENABLE_SYCL` to `ON` depending on the selected backend. If RAJA is installed in a nonstandard location you will be prompted to set the variable `RAJA_DIR` with the path to the RAJA CMake configuration file. To enable building the RAJA examples set `EXAMPLES_ENABLE_CXX` to `ON`.

SUNDIALS has been tested with RAJA version 0.14.0.

1.1.4.11 Building with XBraid

XBraid is parallel-in-time library implementing an optimal-scaling multigrid reduction in time (MGRIT) solver. The library is developed by Lawrence Livermore National Laboratory and is available from the XBraid GitHub repository.

To enable XBraid support, set `ENABLE_XBRAID` to `ON`, set `XBRAID_DIR` to the root install location of XBraid or the location of the clone of the XBraid repository.

Note: At this time the XBraid types `braid_Int` and `braid_Real` are hard-coded to `int` and `double` respectively. As such SUNDIALS must be configured with `SUNDIALS_INDEX_SIZE` set to `32` and `SUNDIALS_PRECISION` set to `double`. Additionally, SUNDIALS must be configured with `ENABLE_MPI` set to `ON`.

SUNDIALS has been tested with XBraid version 3.0.0.

1.1.5 Testing the build and installation

If SUNDIALS was configured with `EXAMPLES_ENABLE_<language>` options to `ON`, then a set of regression tests can be run after building with the `make` command by running:

```
% make test
```

Additionally, if `EXAMPLES_INSTALL` was also set to `ON`, then a set of smoke tests can be run after installing with the `make install` command by running:

```
% make test_install
```

1.1.6 Building and Running Examples

Each of the SUNDIALS solvers is distributed with a set of examples demonstrating basic usage. To build and install the examples, set at least of the `EXAMPLES_ENABLE_<language>` options to `ON`, and set `EXAMPLES_INSTALL` to `ON`. Specify the installation path for the examples with the variable `EXAMPLES_INSTALL_PATH`. CMake will generate `CMakeLists.txt` configuration files (and `Makefile` files if on Linux/Unix) that reference the *installed* SUNDIALS headers and libraries.

Either the `CMakeLists.txt` file or the traditional `Makefile` may be used to build the examples as well as serve as a template for creating user developed solutions. To use the supplied `Makefile` simply run `make` to compile and generate the executables. To use CMake from within the installed example directory, run `cmake` (or `ccmake` or `cmake-gui` to use the GUI) followed by `make` to compile the example code. Note that if CMake is used, it will overwrite the traditional `Makefile` with a new CMake-generated `Makefile`.

The resulting output from running the examples can be compared with example output bundled in the SUNDIALS distribution.

Note: There will potentially be differences in the output due to machine architecture, compiler versions, use of third party libraries etc.

1.1.7 Configuring, building, and installing on Windows

CMake can also be used to build SUNDIALS on Windows. To build SUNDIALS for use with Visual Studio the following steps should be performed:

1. Unzip the downloaded tar file(s) into a directory. This will be the `SOLVERDIR`

2. Create a separate `BUILDDIR`

3. Open a Visual Studio Command Prompt and cd to `BUILDDIR`

4. Run `cmake-gui ../SOLVERDIR`

   a. Hit Configure

   b. Check/Uncheck solvers to be built

   c. Change `CMAKE_INSTALL_PREFIX` to `INSTDIR`

   d. Set other options as desired

   e. Hit Generate

5. Back in the VS Command Window:

   a. Run `msbuild ALL_BUILD.vcxproj`

   b. Run `msbuild INSTALL.vcxproj`

The resulting libraries will be in the `INSTDIR`.

The SUNDIALS project can also now be opened in Visual Studio. Double click on the `ALL_BUILD.vcxproj` file to open the project. Build the whole *solution* to create the SUNDIALS libraries. To use the SUNDIALS libraries in your own projects, you must set the include directories for your project, add the SUNDIALS libraries to your project solution, and set the SUNDIALS libraries as dependencies for your project.

1.2 Installed libraries and exported header files

Using the CMake SUNDIALS build system, the command

```
$ make install
```

will install the libraries under `LIBDIR` and the public header files under `INCLUDEDIR`. The values for these directories are `INSTDIR/lib` and `INSTDIR/include`, respectively. The location can be changed by setting the CMake variable `CMAKE_INSTALL_PREFIX`. Although all installed libraries reside under `LIBDIR/lib`, the public header files are further organized into subdirectories under `INCLUDEDIR/include`.

The installed libraries and exported header files are listed for reference in the table below. The file extension `.LIB` is typically `.so` for shared libraries and `.a` for static libraries. Note that, in this table names are relative to `LIBDIR` for libraries and to `INCLUDEDIR` for header files.

A typical user program need not explicitly include any of the shared SUNDIALS header files from under the `INCLUDEDIR/include/sundials` directory since they are explicitly included by the appropriate solver header files (e.g., `sunlinsol_dense.h` includes `sundials_dense.h`). However, it is both legal and safe to do so, and would be useful, for example, if the functions declared in `sundials_dense.h` are to be used in building a preconditioner.

1.2.1 Using SUNDIALS as a Third Party Library in other CMake Projects

The `make install` command will also install a CMake package configuration file that other CMake projects can load to get all the information needed to build against SUNDIALS. In the consuming project's CMake code, the `find_package` command may be used to search for the configuration file, which will be installed to `instdir/SUNDIALS_INSTALL_CMAKEDIR/SUNDIALSConfig.cmake` alongside a package version file `instdir/SUNDIALS_INSTALL_CMAKEDIR/SUNDIALSConfigVersion.cmake`. Together these files contain all the information the consuming project needs to use SUNDIALS, including exported CMake targets. The SUNDIALS exported CMake targets follow the same naming convention as the generated library binaries, e.g. the exported target for CVODE is `SUNDIALS::cvode`. The CMake code snipped below shows how a consuming project might leverage the SUNDIALS package configuration file to build against SUNDIALS in their own CMake project.

```
project(MyProject)

# Set the variable SUNDIALS_DIR to the SUNDIALS instdir.
# When using the cmake CLI command, this can be done like so:
#   cmake -D SUNDIALS_DIR=/path/to/sundials/installation

find_package(SUNDIALS REQUIRED)

add_executable(myexec main.c)

# Link to SUNDIALS libraries through the exported targets.
# This is just an example, users should link to the targets appropriate
# for their use case.
target_link_libraries(myexec PUBLIC SUNDIALS::cvode SUNDIALS::nvecpetsc)
```

Table 1.1: SUNDIALS shared libraries and header files

Shared	Headers	sundials/sundials_band.h
		sundials/sundials_config.h
		sundials/sundials_context.h
		sundials/sundials_cuda_policies.hpp
		sundials/sundials_dense.h
		sundials/sundials_direct.h
		sundials/sundials_hip_policies.hpp
		sundials/sundials_iterative.h
		sundials/sundials_linearsolver.h
		sundials/sundials_math.h
		sundials/sundials_matrix.h
		sundials/sundials_memory.h
		sundials/sundials_mpi_types.h
		sundials/sundials_nonlinearsolver.h
		sundials/sundials_nvector.h
		sundials/sundials_types.h
		sundials/sundials_version.h
		sundials/sundials_xbraid.h
NVECTOR Modules		
SERIAL	Libraries	libsundials_nvecserial.LIB
	Headers	nvector/nvector_serial.h
PARALLEL	Libraries	libsundials_nvecparallel.LIB
	Headers	nvector/nvector_parallel.h
OPENMP	Libraries	libsundials_nvecopenmp.LIB

Table 1.1 – continued from previous page

	Headers	nvector/nvector_openmp.h
PTHREADS	Libraries	libsundials_nvecpthreads.LIB
	Headers	nvector/nvector_pthreads.h
PARHYP	Libraries	libsundials_nvecparhyp.LIB
	Headers	nvector/nvector_parhyp.h
PETSC	Libraries	libsundials_nvecpetsc.LIB
	Headers	nvector/nvector_petsc.h
CUDA	Libraries	libsundials_nveccuda.LIB
	Headers	nvector/nvector_cuda.h
HIP	Libraries	libsundials_nvechip.LIB
	Headers	nvector/nvector_hip.h
RAJA	Libraries	libsundials_nveccudaraja.LIB
		libsundials_nvechipraja.LIB
	Headers	nvector/nvector_raja.h
SYCL	Libraries	libsundials_nvecsycl.LIB
	Headers	nvector/nvector_sycl.h
MANYVECTOR	Libraries	libsundials_nvecmanyvector.LIB
	Headers	nvector/nvector_manyvector.h
MPIMANYVECTOR	Libraries	libsundials_nvecmpimanyvector.LIB
	Headers	nvector/nvector_mpimanyvector.h
MPIPLUSX	Libraries	libsundials_nvecmpiplusx.LIB
	Headers	nvector/nvector_mpiplusx.h
SUNMATRIX Modules		
BAND	Libraries	libsundials_sunmatrixband.LIB
	Headers	sunmatrix/sunmatrix_band.h
CUSPARSE	Libraries	libsundials_sunmatrixcusparse.LIB
	Headers	sunmatrix/sunmatrix_cusparse.h
DENSE	Libraries	libsundials_sunmatrixdense.LIB
	Headers	sunmatrix/sunmatrix_dense.h
MAGMADENSE	Libraries	libsundials_sunmatrixmagmadense.LIB
	Headers	sunmatrix/sunmatrix_magmadense.h
ONEMKLDENSE	Libraries	libsundials_sunmatrixonemkldense.LIB
	Headers	sunmatrix/sunmatrix_onemkldense.h
SPARSE	Libraries	libsundials_sunmatrixsparse.LIB
	Headers	sunmatrix/sunmatrix_sparse.h
SLUNRLOC	Libraries	libsundials_sunmatrixslunrloc.LIB
	Headers	sunmatrix/sunmatrix_slunrloc.h
SUNLINSOL Modules		
BAND	Libraries	libsundials_sunlinsolband.LIB
	Headers	sunlinsol/sunlinsol_band.h
CUSOLVERSP_BATCHQR	Libraries	libsundials_sunlinsolcusolversp.LIB
	Headers	sunlinsol/sunlinsol_cusolversp_batchqr.h
DENSE	Libraries	libsundials_sunlinsoldense.LIB
	Headers	sunlinsol/sunlinsol_dense.h
KLU	Libraries	libsundials_sunlinsolklu.LIB
	Headers	sunlinsol/sunlinsol_klu.h
LAPACKBAND	Libraries	libsundials_sunlinsollapackband.LIB
	Headers	sunlinsol/sunlinsol_lapackband.h
LAPACKDENSE	Libraries	libsundials_sunlinsollapackdense.LIB
	Headers	sunlinsol/sunlinsol_lapackdense.h
MAGMADENSE	Libraries	libsundials_sunlinsolmagmadense.LIB

continues on next page

Table 1.1 – continued from previous page

	Headers	sunlinsol/sunlinsol_magmadense.h
ONEMKLDENSE	Libraries	libsundials_sunlinsolonemkldense.LIB
	Headers	sunlinsol/sunlinsol_onemkldense.h
PCG	Libraries	libsundials_sunlinsolpcg.LIB
	Headers	sunlinsol/sunlinsol_pcg.h
SPBCGS	Libraries	libsundials_sunlinsolspbcgs.LIB
	Headers	sunlinsol/sunlinsol_spbcgs.h
SPFGMR	Libraries	libsundials_sunlinsolspfgmr.LIB
	Headers	sunlinsol/sunlinsol_spfgmr.h
SPGMR	Libraries	libsundials_sunlinsolspgmr.LIB
	Headers	sunlinsol/sunlinsol_spgmr.h
SPTFQMR	Libraries	libsundials_sunlinsolsptfqmr.LIB
	Headers	sunlinsol/sunlinsol_sptfqmr.h
SUPERLUDIST	Libraries	libsundials_sunlinsolsuperludist.LIB
	Headers	sunlinsol/sunlinsol_superludist.h
SUPERLUMT	Libraries	libsundials_sunlinsolsuperlumt.LIB
	Headers	sunlinsol/sunlinsol_superlumt.h
SUNNONLINSOL Modules		
NEWTON	Libraries	libsundials_sunnonlinsolnewton.LIB
	Headers	sunnonlinsol/sunnonlinsol_newton.h
FIXEDPOINT	Libraries	libsundials_sunnonlinsolfixedpoint.LIB
	Headers	sunnonlinsol/sunnonlinsol_fixedpoint.h
PETSCSNES	Libraries	libsundials_sunnonlinsolpetscsnes.LIB
	Headers	sunnonlinsol/sunnonlinsol_petscsnes.h
SUNMEMORY Modules		
SYSTEM	Libraries	libsundials_sunmemsys.LIB
	Headers	sunmemory/sunmemory_system.h
CUDA	Libraries	libsundials_sunmemcuda.LIB
	Headers	sunmemory/sunmemory_cuda.h
HIP	Libraries	libsundials_sunmemhip.LIB
	Headers	sunmemory/sunmemory_hip.h
SYCL	Libraries	libsundials_sunmemsycl.LIB
	Headers	sunmemory/sunmemory_sycl.h
SUNDIALS Packages		
CVODE	Libraries	libsundials_cvode.LIB
	Headers	cvode/cvode.h
		cvode/cvode_bandpre.h
		cvode/cvode_bbdpre.h
		cvode/cvode_diag.h
		cvode/cvode_direct.h
		cvode/cvode_impl.h
		cvode/cvode_ls.h
		cvode/cvode_proj.h
		cvode/cvode_spils.h
CVODES	Libraries	libsundials_cvodes.LIB
	Headers	cvodes/cvodes.h
		cvodes/cvodes_bandpre.h
		cvodes/cvodes_bbdpre.h
		cvodes/cvodes_diag.h
		cvodes/cvodes_direct.h
		cvodes/cvodes_impl.h
		cvodes/cvodes_ls.h

continues on next page

Table 1.1 – continued from previous page

		cvodes/cvodes_spils.h
ARKODE	Libraries	libsundials_arkode.LIB
		libsundials_xbraid.LIB
	Headers	arkode/arkode.h
		arkode/arkode_arkstep.h
		arkode/arkode_bandpre.h
		arkode/arkode_bbdpre.h
		arkode/arkode_butcher.h
		arkode/arkode_butcher_dirk.h
		arkode/arkode_butcher_erk.h
		arkode/arkode_erkstep.h
		arkode/arkode_impl.h
		arkode/arkode_ls.h
		arkode/arkode_mristep.h
		arkode/arkode_xbraid.h
IDA	Libraries	libsundials_ida.LIB
	Headers	ida/ida.h
		ida/ida_bbdpre.h
		ida/ida_direct.h
		ida/ida_impl.h
		ida/ida_ls.h
		ida/ida_spils.h
IDAS	Libraries	libsundials_idas.LIB
	Headers	idas/idas.h
		idas/idas_bbdpre.h
		idas/idas_direct.h
		idas/idas_impl.h
		idas/idas_spils.h
KINSOL	Libraries	libsundials_kinsol.LIB
	Headers	kinsol/kinsol.h
		kinsol/kinsol_bbdpre.h
		kinsol/kinsol_direct.h
		kinsol/kinsol_impl.h
		kinsol/kinsol_ls.h
		kinsol/kinsol_spils.h

Index