Text Parser Library

User's Manual

June 2014

Contents

Chapter 1

About this document

This document is the user manual for Text Parser library.

1.1 About Text Parser Library

Text Parser library describes and obtains the parameters of simulation software programmed in C/C++ and/or Fortran. It reads the parameters in an ASCII file specified by the user, retains the parameters in Text Parser library itself, and enables the user to obtain the parameters through a simulator. Additionally, it operates through many kinds of simulation programs on multiple platforms.

1.2 Format

The following format represents a Shell command:

```
$ command (command parameter)
```

or,

```
# command (command parameter)
```

A command that starts with "$" is to be executed by a user. A command that starts with "#" is to be executed by the administrator (root user in most cases).

1.3 Supported Environments

Text Parser library supports the following environments:

- GNU/Linux
  - CentOS
    OS CentOS6.2 i386/x86_64
    gcc/g++/gfortran(Fortran90) gcc version 4.4.5
  - Debian GNU/Linux
    OS Debian 6 squeeze i386/amd64
    gcc/g++/gfortran(Fortran90) gcc version 4.4.5
- MacOS X Snow Leopard
  - OS MacOS X Snow Leopard
  - gcc/g++/gfortran(Fortran90) gcc4.4 and its development package
- MacOS X Lion
  - OS MacOS X Lion
  - gcc/g++/gfortran(Fortran90) gcc4.4 and its development package
- Microsoft Windows
  - Windows7(64bit)
    OS Windows7(64bit) Cygwin 1.7.9

    gcc/g++/gfortran(Fortran90) 4.5
  - WindowsXP(32bit)
    OS WindowsXP(32bit) Cygwin 1.7.9
    gcc/g++/gfortran(Fortran90) 4.5

MPI version supports the following environment:

- GNU/Linux
  - CentOS
    OS CentOS6.2 x86_64
    gcc/g++/gfortran(Fortran90) gcc version 4.4.5
    MPICH openMPI 1.4.4

Chapter 2

Building Text Parser Library

2.1 Package Structure

The Text Parser library package is stored in a file with the following name format:

`TextParser-x.x.tar.gz` (where `x.x.` designates the version.)

When the file is expanded, the following directories and files should appear:

```
TextParser-x.x
   ├────doc
   ├────Examples
   ├────include
   ├────lib
   ├────m4
   └────src
```

doc/ Contains all Text Parser library documents including this user guide.
Examples/ Contains example libraries and examples of input files for testing.
include/ Contains header files. The contents of this directory will be installed in `$prefix/include` after a `make install` is invoked.
m4/ Contains macros for autotools.
src/ Contains source files.
lib/ Libraries will be created and installed in `$prefix/lib` after a `make install` is invoked.

2.2 Building the Text Parser Library Package

Text Parser library offers two types of package builds: an environment configuration with configure script, and a manual environment configuration. The following guide describes how to first build a package using a configure script, and then with a manual environment configuration. For a description on how to build a package manually, skip to Section 2.8.

2.3 Building using a Configure Script

Use a shell environment to build a package. The syntax for setting the environmental variables differs according to the shell used. Replace the environmental variables settings in this section according to the computer environment used. For a Windows Cygwin environment, the Fortran compiler must be specified with the configure script. For further details, see Section 2.7. For a description on how to build an MPI version, skip to Section 2.6.

In the following example, a working directory is created, and an unzipped package is built and installed into this working directory.

1. Make a working directory (called "work" herein) into which the Text Parser library package can be copied.

   ```
   $ mkdir work
   $ cp [package path] work
   ```

2. Change to the working directory and unzip the package.

   ```
   $ cd work
   $ tar zxf TextParser-x.x.tar.gz
   ```

3. Change to the TextParser-X.Y directory generated by unzipping.

   ```
   $ cd TextParser-X.Y
   ```

4. Execute the configure script, specifying the appropriate configuration options.

   ```
   $ ./configure
   ```

   The configure script can be configured according to the environment used by choosing the appropriate option. For further details on the options available, see Section 2.4. A Makefile suitable for the environment used will be created in the directory by executing the configure script with the appropriate options.

5. Execute a make command to create the libraries and build the test programs.

   ```
   $ make
   ```

   This will create the following files:
   src/libTP.a
   src/libTPmpi.a
   src/libTP_fapi.a

   The libTPmpi.a will be created only if the MPI version option is enabled and the libTP_fapi.a will be created only if the Fortran build option is enabled.

   If you want to re-build Text Parser library, delete the files created by the previous make command using the following command:

   ```
   $ make clean
   ```

If you want to rerun the configure script or recreate the makefile, execute a `make distclean` to delete all information and restart from the configure script.

```
$ make distclean
```

6. Execute a `make install` to install the libraries and the header files into the directory specified with a `--prefix` option for the configure script. The directory and files to install are as follows:

```
${prefix}
   ├──────lib
   │        ├──────libTP.a
   │        ├──────libTPmpi.a
   │        └──────libTP_fapi.a
   └──────include
            ├──────TextParser.h
            ├──────TextParserCommon.h
            ├──────TextParserElement.h
            ├──────TextParserTree.h
            └──────TextParser.inc
```

Here, X, Y, and Z represent the version number.
The libTPmpi.a will be created only if the MPI version option is enabled and the libTP_fapi.a will be created only if the Fortran build option is enabled.

For further details on how to configure the installation directory under a `prefix`, see Section 2.4.
If you have permission to write under the installation directory, execute the following command:

```
$ make install
```

If administrator rights are required to write under the installation directory, use the `sudo` command as follows.

```
$ sudo make install
```

If administrator rights are required to write under the installation directory and you cannot use `sudo`, log in as a root user and execute a `make install`.

```
$ su
passward:
# make install
# exit
```

The uninstall command is different depending on your permissions:
`$ make uninstall` ,
`$ sudo make uninstall` , or,
`# make uninstall` .

2.4 Configure Script Options

Specify the install directory, compiler, and MPI support option as follows:

- `--prefix`=dir
  Specifies where to install the package. When `--prefix`=/usr/local is specified, the libraries and the header files will be installed under the following directories:

  Library: /usr/local/lib
  Header file: /usr/local/include
  If this option is not specified, /usr/local/ is used as the default value for the installation.

- `--enable-fapi`
  This toggles between building a Fortran API library and Fortran test programs.
  Specify `--enable-fapi` to enable this option. This option is disabled by default.
- Compiler and linker options
  Compilers, linkers, and their options are found semi-automatically. If you want to use nonstandard commands or libraries, you must specify them with the following configure script options:

  `CC` Command path of the C compiler.

  `CFLAGS` Flag to pass to the C compiler.

  `CXX` Command path of the C++ compiler.

  `CXXFLAGS` Flag to pass to the C++ compiler.

  `LDFLAGS` Flag to pass to the linker. For example, if the library is in a nonstandard location`<libdir>`, you can specify it using: `-L<libdir>`.

  `LIBS` Flag to pass a library for the linker. For example, if you want to use a library `<library>`, specify it as: `-l<library>`.

  `CPP` Command path of the preprocessor.

  `CPPFLAGS` Flag to pass to the preprocessor. For example, if you want to use a header file in a nonstandard directory, i.e., `<include dir>`, specify it as `-I<include dir>`.

  `F77` Command path of the Fortran 77 compiler.

  `FFLAGS` Flag to pass to the Fortran 77 compiler.

  `FC` Command path of the Fortran compiler.

  `FCFLAGS` Flag to pass to the Fortran compiler.

  `CXXCPP` Command path of the C++ preprocessor.

The following command is an example of specifying /usr/local/TextParser as a prefix and gfortran as a Fortran compiler:

```
$ ./configure --prefix=/usr/local/TextParser FC=gfortran
```

If `$ ./configure --help` is executed, other options will be displayed. Note that valid options are simply installation directory specifications such as `--prefix, --includedir, --libdir, and --enable-fapi` along with the compile options described above.

2.5 Building Fortran Version

To build a Fortran API library and Fortran sample programs, specify the option for the configure script (for further details, see Section 2.4.). In addition, you might have to specify a Fortran compiler for the FC according to the environment you are using. In the following example option, Fortran is enabled and `gfortran` is specified for the Fortran compiler:

```
$ ./configure FC=gfortran --enable-fapi
```

If this option is enabled, libTP_fapi.a will be installed and `-lTP_fapi` can be specified for linking. When specifying `-lTP_fapi` for linking, specify `-lTPmpi` or `-lTP` as the link option.

2.6 Building MPI Version

To build a C++ library for the MPI version, specify the C++ compiler for the MPI version for CXX, the option for the configure script.

In the following example, mpicc is specified as the C++ compiler.

```
$ ./configure CXX=mpicc
```

If any compile or link options must be specified based on the compiler used, specify them using `CXXFLAGS` or `LDFLAGS`. If this option is enabled, libTPmpi.a will be installed and `-lTPmpi` can be specified for linking. Be sure NOT to use the following two link options together: `-lTPmpi` and `-lTP`.

If the MPI version is enabled, the `ENABLE_MPI` macro will be defined in config.h generated during the configure script execution. To use this macro in a user program, include config.h in the target user program. Note that config.h will not be installed under the installation directory that you specified with `$prefix` (where the header files such as TextParser.h are installed) because config.h is provided mainly for storing the configuration when Text Parser library is built.

2.7 Building and Using Text Parser Library in a Windows Cygwin Environment

Text Parser library is available for a Cygwin 1.7.9 environment. However, the building will fail if a Fortran compiler is not specified for the configure script because the Fortran compiler, gcc v3 base, provided by default, is not suitable for a Cygwin environment. Be sure to specify gfortran (gcc v4 base) as the Fortran compiler for a Cygwin environment. Specify the gfortran (gcc v4 base) as the Fortran compiler as follows:

```
$ ./configure FC=gfortran F77=gfortran
```

2.8 Manually Building Text Parser Library

This section and the following subsections describe how to build Text Parser library manually. If Text Parser library is built manually, the following static library files will be generated. To utilize the library, the following files are required, and should be copied into the proper directory.

Library files:
```
lib/libTP.a (Normal version)
lib/libTPmpi.a (MPI version)
lib/libTP_fapi.a (API for FORTRAN)
```

Include files:
```
include/TextParser.h   (for C/C++)
include/TextParserCommon.h
include/TextParserTree.h
include/TextParserElement.h
include/TextParser.inc (for FORTRAN)
```

2.8.1 Editing the Makefile for Manual Configuration

Makefile_hand, a file used for manual configuration, is located in the top/, src/ and Examples/ directories. Use these directories for a manual configuration.

Modify Makefile_hand in the top/ directory according to the system and environment used. In a distributed example, the variables are modified as follows, and this variable settings are reflected to both the lib/ and Examples/ directories.

- C++ compiler for mpi `MPICXX = mpicxx`
- Fortran compiler `FC = gfortran`

If you are planning to build MPI libraries and usage examples, make sure to configure `MPICXX`. Variables other than `MPICXX` are based on the default values of make, and thus `CXX, CXXLAGS, AR, ARFLAGS, and RANLIB` are available.

If `MPICXX` is not configured, it will be replaced by the default value of `make`. In the distributed example, a normal C++ code uses the default C++ compiler for `make`. If you want to use a compiler or a command other than the default settings, a configuration is required.

If Text Parser library is built using a manual configuration, static library (.a) files will be created under the lib/ directory.

2.8.2 Make (default)

In the top/ directory of the package, execute a make as follows:

```
$ make -f Makefile_hand
```

This will create the following files:

lib/ directory:
`lib/libTP.a`

Examples/ directory:

```
Examples/Example1_cpp
Examples/Example2_cpp
Examples/Example3_cpp
Examples/Example4_cpp
Examples/Example5_cpp
Examples/Example6_cpp
Examples/Example7_cpp

Examples/Example1_c
Examples/Example2_c
Examples/Example3_c
Examples/Example4_c
Examples/Example5_c
Examples/Example7_c
```

2.8.3 Make (MPI Version)

In the top/ directory of the package, execute a make as follows:

```
$ make -f Makefile_hand mpi
```

This will create the following files:

lib/ directory:
`lib/libTPmpi.a`

Examples/ directory
`Examples/Example3_cpp_mpi`

2.8.4 Make (API for Fortran)

In the top/ directory of the package, execute a make as follows:

```
$ make -f Makefile_hand fapi
```

This will create the following files:

lib/ directory:
```
lib/libTP_fapi.a
```

Examples/ directory
```
Examples/Example1_f90
Examples/Example2_f90
Examples/Example3_f90
Examples/Example4_f90
Examples/Example5_f90
Examples/Example7_f90
```

Chapter 3

Using Text Parser Library - Building and Execution -

Text Parser library is available in C++, C, and Fortran90 programs. This chapter describes how to build user programs using Text Parser library. The following section examples assume that Text Parser library was built in the following manner:

- "`prefix=/usr/local/TextParser`" was specified for the configure script,
- libraries were installed in `/usr/local/TextParser/lib`, and
- header files were installed in `/usr/local/TextParser/include`.

3.1 C++

To compile and link mymain.cpp, a C++ program using Text Parser library, with g++, run the following commands:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/TextParser/lib
$ g++ -o mymain mymain.cpp -I/usr/local/TextParser/include \
 -L/usr/local/TextParser/lib -lTP
```

Be sure NOT to use the following two options together: -lTPmpi (the option for the MPI version) and -lTP (the option for a normal version).

3.2 C

To compile and link mymain.c, a C program using Text Parser library, with gcc, run the following commands:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/TextParser/lib
$ gcc -o mymain mymain.c -I/usr/local/TextParser/include \
-lstdc++ -L/usr/local/TextParser/lib -lTP
```

Be sure NOT to use the following two options together: -lTPmpi (the option for the MPI version) and -lTP (the option for a normal version).

3.3 Fortran 90

To compile mymain.f90, a Fortran90 program using Text Parser library, with gfortran, run the following commands:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/TextParser/lib
$ gfortran -o mymain mymain.c -I/usr/local/TextParser/include \
-lstdc++ -L/usr/local/TextParser/lib -lTP -lTP_fapi
```

Be sure NOT to use the following two options together: -lTPmpi (the option for the MPI version) and -lTP (the option for a normal version). When building Text Parser library, be sure to specify `''--enable-fapi''` for the configure script to enable libTP_fapi.a.

3.4 Adding a Library Path to an Environmental Variable

To execute an executable file to which a shared library is linked, the library path must be specified. To specify a library path, add the path to the environmental variable, "LD_LIBRARY_PATH." For instance, if a library is installed in /usr/local/TextParser/lib, edit it as follows:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/TextParser/lib
```

3.5 How to Use Text Parser Library in MPI Parallel Programs

The steps involved when using Text Parser library in user MPI parallel programs are as follows:

1. Build and install the MPI version library. For further details, see Sections 2.2,2.4,and 2.6.
2. Specify LD_LIBRARY_PATH.
3. Build the program in an MPI parallel environment.

The following is an example of compiling a user program, mymain.cpp, with mpicxx in an environment where Text Parser library was installed in /usr/local/TextParser/lib:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/TextParser/lib
$ mpicxx -o mymain mymain.cpp -I/usr/local/TextParser/include \
 -L/usr/local/TextParser/lib -lTPmpi
```

Be sure NOT to use the following two options together: -lTPmpi (the option for the MPI version) and -lTP (the option for a normal version).

If you want to compute this user program in four parallel environments and use `mpirun` as the MPI execution command, execute the program as follows:

```
$ mpirun -np 4  mymain
```

Chapter 4

Using Text Parser Library for User Programs

The following sections describe Text Parser library API. The functional groups of the API are defined in two header files: TextParser.h for C++ and C, and TextParser.inc for Fortran 90. Include these files if you want to utilize the functional groups. TextParser.h and TextParser.inc are installed under $prefix/include after running a make install. This $prefix is the directory you specified during the configure script execution.

4.1 Programs in Examples/ Directory

In the Examples/ directory, source codes are provided as usage examples. Please feel free to use them as a reference.

4.1.1 C++ Examples

• Example1_cpp.cpp	Input/output of a parameter file and discarding of data
• Example2_cpp.cpp	Input of a parser file that will generate an error or warning
• Example3_cpp.cpp	Obtaining all parameters (full path)
• Example4_cpp.cpp	Obtaining all parameters (relative path)
• Example5_cpp.cpp	Rewriting and deleting the VALUE of Leaf, and creating a new Leaf VALUE
• Example6_cpp.cpp	Check on a value with dependency
• Example7_cpp.cpp	Test for @range and @list
• Example7_cpp.cpp	Test for @range and @list (that will generate an error)

These programs are built during the make execution, and the following executable files are then created: ./Example1_cpp, ./Example2_cpp, ./Example3_cpp, ./Example4_cpp, /Example5_cpp and ./Example6_cpp.

4.1.2 C Examples

• Example1_c.c	Input/output of a parameter file and discarding of data
• Example2_c.c	Input of a parser file that will generate an error or warning
• Example3_c.c	Obtaining all parameters (full path)
• Example4_c.c	Obtaining all parameters (relative path)
• Example5_c.c	Rewriting and deleting the VALUE of Leaf, and creating a new Leaf VALUE
• Example7_c.c	Test for @range and @list

These programs are built during the make execution, and the following executable files are then created: ./Example1_c, ./Example2_c, ./Example3_c, ./Example4_c, and ./Example5_c.

4.1.3 Fortran Examples

• Example1_f90.f90	Input/output of a parameter file and discarding of data
• Example2_f90.f90	Input of a parser file that will generate an error or warning
• Example3_f90.f90	Obtaining all parameters (full path)
• Example4_f90.f90	Obtaining all parameters (relative path)
• Example5_f90.f90	Rewriting and deleting VALUE of Leaf, and creating a new Leaf VALUE
• Example7_f90.f90	Test for @range and @list

These programs are built during a make execution, and the following executable files are then created: ./Example1_f90, ./Example2_f90, ./Example3_f90, and ./Example4_f90.

4.1.4 Examples of C++ MPI Parallel Programs

- Example3_cpp_mpi.cpp Obtaining all parameters (full path)
- Example3-1_cpp_mpi.cpp Reading different files according to their rank.

These programs are built during a make execution, and ./Example3_cpp_mpi and ./Example3-1_cpp_mpi are then created. If the MPI version was enabled when building Text Parser library, the process of rank 0 will read a file and send the contents of the file to all processes; all processes will then parse and store the content data. If you want to compute your program (mymain) in four parallel environments and use `mpirun` as the MPI execution command, execute the program as follows:

```
$ mpirun -np 4  mymain
```

4.2 Usage in C++ Programs

To use Text Parser library in C++ programs, include TextParser.h, which provides a TextParser class that organizes the API to allow users to access Text Parser library. Use the methods of the TextParser class to utilize Text Parser library in your programs.

4.2.1 Include File

When using Text Parser library in a user program, simply include TextParser.h. You do not have to include any other files because TextParser.h will read the other necessary files. The data types to be used in a user program are defined in TextParserCommon.h

4.2.2 TextParserValueType

TextParserValueType is defined in TextParserCommon.h as shown in Table 4.1.

Table. 4.1 TextParserValueType

TextParserValueType defined value	Value type
TP_UNDEFFINED_VALUE = 0	Undefined value
TP_NUMERIC_VALUE = 1	Numerical value
TP_STRING_VALUE = 2	String
TP_DEPENDENCE_VALUE = 3	Value with dependency
TP_VECTOR_UNDEFFINED = 4	Vector of undefined value
TP_VECTOR_NUMERIC = 5	Vector of numerical value
TP_VECTOR_STRING = 6	Vector of strings

After a leaf is obtained, the value type indicated by its leaf path can be used to judge the conditions for the next conversion process.

4.2.3 TextParserError

TextParserError is defined in TextParserCommon.h as shown in Tables 4.2 and 4.3.

Table. 4.2 TextParserError 1

TextParserError defined value	Value type
TP_NO_ERROR = 0	No error
TP_ERROR = 100	Error
TP_DATABASE_NOT_READY_ERROR = 101	Cannot access the database
TP_DATABASE_ALREADY_SET_ERROR = 102	Database already read
TP_FILEOPEN_ERROR = 103	File open error
TP_FILEINPUT_ERROR = 104	File input error
TP_FILEOUTPUT_ERROR = 105	File output error
TP_ENDOF_FILE_ERROR = 106	Reached the end of the file
TP_ILLEGAL_TOKEN_ERROR = 107	Illegal token
TP_MISSING_LABEL_ERROR = 108	Label not found
TP_ILLEGAL_LABEL_ERROR = 109	Illegal label
TP_ILLEGAL_ARRAY_LABEL_ERROR = 110	Illegal array type label
TP_MISSING_ELEMENT_ERROR = 111	Element not found
TP_ILLEGAL_ELEMENT_ERROR = 112	Illegal element
TP_NODE_END_ERROR = 113	Too many node end
TP_NODE_END_MISSING_ERROR = 114	Node end not found
TP_NODE_NOT_FOUND_ERROR = 115	Node not found
TP_LABEL_ALREADY_USED_ERROR = 116	label already used
TP_LABEL_ALREADY_USED_PATH_ERROR = 117	Label already used in the path
TP_ILLEGAL_CURRENT_ELEMENT_ERROR = 118	Illegal current element
TP_ILLEGAL_PATH_ELEMENT_ERROR = 119	Illegal path element
TP_MISSING_PATH_ELEMENT_ERROR = 120	Path element not found
TP_ILLEGAL_LABEL_PATH_ERROR = 121	Illegal label path
TP_UNKNOWN_ELEMENT_ERROR = 122	Unknown element

Table. 4.3 TextParserError 2

TextParserError defined value	Value type
TP_MISSING_EQUAL_NOT_EQUAL_ERROR = 123	Missing == or !=
TP_MISSING_AND_OR_ERROR = 124	Missing && or ||
TP_MISSING_CONDITION_EXPRESSION_ERROR = 125	Missing condition expression
TP_MISSING_CLOSED_BRANCKET_ERROR = 126	Missing closed bracket
TP_ILLEGAL_CONDITION_EXPRESSION_ERROR = 127	Illegal condition expression
TP_ILLEGAL_DEPENDENCE_EXPRESSION_ERROR = 128	Illegal dependence expression
TP_MISSING_VALUE_ERROR = 129	Missing value
TP_ILLEGAL_VALUE_ERROR = 130	Illegal value
TP_ILLEGAL_NUMERIC_VALUE_ERROR = 131	Illegal numeric value
TP_ILLEGAL_VALUE_TYPE_ERROR = 132	Illegal value type of a vector
TP_MISSING_VECTOR_END_ERROR = 133	Missing vector end
TP_VALUE_CONVERSION_ERROR = 134	Value conversion error
TP_MEMORY_ALLOCATION_ERROR = 135	Memory allocation error
TP_REMOVE_ELEMENT_ERROR = 136	Remove element error
TP_MISSING_COMMENT_END_ERROR = 137	Missing comment end
TP_ID_OVER_ELEMENT_NUMBER_ERROR = 138	ID exceeds the element number
TP_GET_PARAMETER_ERROR = 139	Parameter obtainment
TP_UNSUPPORTED_ERROR = 199	Unsupported
TP_WARNING = 200	Error
TP_UNDEFINED_VALUE_USED_WARNING = 201	Undefined value used
TP_UNRESOLVED_LABEL_USED_WARNING = 202	Unresolved label used

4.2.4 Obtaining an Instance

The TextParser class involves two types of instances: the only instance generated by a singleton pattern, and an instance that can be created by a user as many times as needed.

■Method for obtaining a singleton instance The method used to obtain the pointer to a singleton instance is defined in TextParser.h as follows:

Creating an instance and obtaining a pointer to the instance

```
static TextParser* TextParser::get_instance_singleton();
```

Return Value pointer to the singleton instance of TextParser class

■Method for obtaining an instance Users can create an instance by using a constructor. For example, an instance can be created as follows:

```
TextParser tp_instance ;
TextParser *  tp_ptr= new TextParser;
```

Each member function can be accessed from a user program by using the pointer of a singleton instance or the pointer to an arbitrarily created instance.

4.2.5 File I/O and Memory Release

This subsection describes the following functions: two functions that read a parameter file and store the analysis results of this file in the data structure, and a function that outputs all parameters stored in the data structure:

Reading from a parameter file

```
TextParserError TextParser::read(const std::string& file);
```

`file` Input file name
Return value error code (=0: no error), which is defined in `TextParseError`.

Reading from a parameter file (MPI local file version)

```
TextParserError TextParser::read_local(const std::string& file);
```

file Input file name
Return value error code (=0: no error), which is defined in `TextParseError`.

Output to a file

```
TextParserError TextParser::write(const std::string& file);
```

`file` Output file name
Return value error code (=0: no error), which is defined in `TextParseError`.

TextParser::read of the MPI version reads a parameter file in the process of rank = 0 and distributes the content of the parameter file to all nodes. Therefore, be sure to locate the parameter file in a directory that can be accessed by the computer where the rank 0 process will run.

With TextParser::read_local, which should be used in the MPI version, each node reads a specified parameter file independently. TextParser::read_local of a non-MPI version simply returns an error and displays an error message.

The data structure to be written in a file is pre-read from another file, analyzed and processed, and thus the conditional expression of its value with dependency is determined when the file reading is complete.

In addition, the function that discards the data of the stored parameters can be defined as follows:

Discarding parameter data

```
TextParserError TextParser::remove();
```

Return value error code (=0: no error), which is defined in `TextParseError`.

4.2.6 Obtaining a Label and a Value (Full Path)

This subsection describes the three functions used to obtain the following: all leaf paths of the parameters stored in the data structure, a parameter value by specifying a path, and the type of parameter value.

Obtaining paths to all parameters

```
TextParserError TextParser::getAllLabels(std::vector<std::string>& labels);
```

`labels` Leaf paths to all parameters
Return value error code (=0: no error), which is defined in `TextParseError`.

Obtaining a value by specifying a path

```
TextParserError TestParser::getValue(const std::string& label,
                                     std::string& value);
```

`label` Path to the parameter
`value` Parameter value
Return value error code (=0: no error), which is defined in `TextParseError`.

Obtaining a value type

```
TextParserValueType getType(const std::string& label, int *error);
```

`label` Path to the parameter
`value` error code (=0: no error), which is defined in `TextParseError`.
Return value type of parameter value. For further details, see `TextParserType` in Table 4.1.

4.2.7 Obtaining a Known Data Type Value (Full Path)

This subsection describes the functions used to obtain the value of a stored parameter whose data type is already known:

Obtaining an integer value

```
bool getInspectedValue(const std::string label, int &ct );
```

`label` Path to the parameter
`ct` For returning the variables (output parameters)
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining a single-precision real number

```
bool getInspectedValue(const std::string label, float &ct );
```

`label` Path to the parameter
`ct` For returning the variables (output parameters)
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining a double-precision real number

```
bool getInspectedValue(const std::string label, double &ct );
```

`label` Path to the parameter
`ct` For returning variables (output parameters)
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining a string value

```
bool getInspectedValue(const std::string label, string &ct );
```

`label` Path to the parameter
`ct` For returning variables (output parameters)
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining the values of an integer vector

```
bool getInspectedVector(const std::string label, int *vec, const int nvec );
```

`label` Path to the parameter
`vec` Array pointer stored in the vector (output parameter)
`nvec` Vector size
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining the values of a single-precision vector

```
bool getInspectedVector(const std::string label, float *vec, const int nvec );
```

`label` Path to the parameter
`vec` Array pointer stored in the vector (output parameter)
`nvec` Vector size
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining the values of a double-precision vector

```
bool getInspectedVector(const std::string label, double *vec, const int nvec );
```

`label` Path to the parameter
`vec` Array pointer stored in the vector (output parameter)
`nvec` Vector size
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

Obtaining the values of a string vector

```
bool getInspectedVector(const std::string label, std::string *vec, const int nvec );
```

`label` Path to the parameter
`vec` Array pointer stored in the vector (output parameter)
`nvec` Vector size
Return value success or failure of the process

Be sure to specify a real index if the label contains an array expression.

4.2.8 Functions for Accessing a Label Relative Path

This subsection describes the following functions: obtaining the current node, obtaining a child node, and changing the nodes:

Obtaining the current node

```
TextParserError TestParser::currentNode(std::string& node);
```

node Path to the current node
Return value error code (=0: no error). For further details, see `TextParseError`.

Changing nodes

```
TextParserError TestParser::changeNode(const std::string& label);
```

label Label of the destination node (relative path)
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining the label of a child node of the current node

```
TextParserError TestParser::getNodes(std::vector<std::string>& labels,
                               int order=0 );
```

labels List of labels of the childe nodes (relative path)
order The output order of the labels is as follows. 0: the order of data storage; 1: the index order of an array label; and 2: the order of appearance in a parameter file.
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining a leaf label of the current node

```
TextParserError TestParser::getLabels(std::vector<std::string> & labels,
                                      int order=0);
```

labels List of leaf labels (relative path)
order The output order of the labels is as follows. 0: the order of the data storage; 1: the index order of an array label; and 2: the order of appearance in a parameter file.
Return value error code (=0: no error). For further details, see `TextParseError`.

4.2.9 Functions for Type Conversion

This subsection describes the functions used to convert a parameter value (string) into a specified type.

Conversion of string value type

```
char TextParser::convertChar(const std::string value, int *error);
short TextParser::convertShort(const std::string value, int *error);
int TextParser::convertInt(const std::string value, int *error);
long TextParser::convertLong(const std::string value, int *error);
long long TextParser::convertLongLong(const std::string value, int *error);
float TextParser::convertFloat(const std::string value, int *error);
double TextParser::convertDouble(const std::string value, int *error);
bool TextParser::convertBool(const std::string value, int *error);
```

value Parameter value (string)
error error code (=0: no error). For further details, see `TextParseError`.
Return value a value of the specified type converted from the parameter value.

As for `convertBool`, it converts the string values, as shown in Table 4.4.

Table. 4.4 Conversion of convertBool

String of value	convertBool Return value (bool)
true	true
TRUE	true
1	true
false	false
FALSE	false
0	false

4.2.10 Splitting a Vector-Type Value

This subsection describes a function for splitting a vector-type parameter value into elements (string):

Obtaining a list of elements of a vector-type parameter

```
TextParserError TestParser::splitVector(const std::string& vector_value,
                                        std::vector<std::string>& velem);
```

vector_value Values of a vector-type parameter (string)
velem Value of each element (string)
Return value error code (=0: no error). For further details, see `TextParseError`.

4.2.11 Editing Parameter Data

This subsection describes the functions for editing the data of the stored parameters:

Deleting the parameter data

```
TextParserError TestParser::deleteLeaf(std::string& label);
```

label Label of a leaf to delete
Return value error code (=0: no error). For further details, see `TextParseError`.

Updating the parameter data

```
TextParserError TestParser::updateValue(std::string& label,std::string& value);
```

label Label of a leaf to update
value Leaf value to update. Be sure to set a string that does not change the type of the leaf (TP_VALUE_TYPE).
Return value error code (=0: no error). For further details, see `TextParseError`.

Creating the parameter data

```
TextParserError TestParser::createLeaf(std::string& label,std::string& value);
```

label Label of a leaf to create
value Value of the leaf to create. If the value is a string, enclose it with double quotation marks.
Return value error code (=0: no error). For further details, see `TextParseError`.

4.2.12 API for Specifying a Range of Parameter Values

Obtaining the parameters by specifying a sequence of numbers with a constant increase or decrease

```
TextParserError splitRange(const std::string & value,
   double *from,double *to,double *step);
```

value Value in a string
from Start value
to End value
step Value with a constant increase or decrease
Return value error code (=0: no error). For further details, see `TextParseError`.

Expanding the parameters by specifying a sequence of numbers with a constant increase or decrease

```
TextParserError expandRange(const std::string & value,
    std::vector<double>& expanded);
```

value Value in a string
expanded Vector of the expanded double values
Return value error code (=0: no error). For further details, see `TextParseError`.

Expanding the parameters by specifying an arbitrary sequence of numbers

```
TextParserError splitList(const std::string & value,std::vector<double>& list,
  TextParserSortOrder order=TP_SORT_NONE);
```

value Value in a string
list Vector of the expanded double values
order Defined in TextParserSortOrder. TP_SORT_NONE: order of description; TP_SORT_ASCENDING: ascending order; TP_SORT_DESCENDING: descending order
Return value error code (=0: no error). For further details, see `TextParseError`.

Table. 4.5 Defined value of TextParserSortOrder

TP_SORT_NONE	no specification, the order of description
TP_SORT_ASCENDING	ascending order
TP_SORT_DESCENDING	descending order

4.2.13 Other API

Checking the existence of a label

```
bool chkLabel(const std::string label);
```

label Label to check (absolute path)
Return value success or failure.

Be sure to specify a real index if the label contains an array expression.

Checking the existence of a node

```
bool chkNode(const std::string label);
```

label Node to check (absolute path)
Return value success or failure.

Be sure to specify a real index if the label contains an array expression.

Obtaining a string of [nnode]th node

```
bool getNodeStr(const std::string label, const int nnode, std::string &ct);
```

label Node (absolute path)
nnode Order of appearance of strings to obtain
ct Obtained string (output parameters)
Return value success or failure.

Be sure to specify a real index if the label contains an array expression.

Counting the number of labels just under the specified node

```
 int countLabels(const std::string label);
```

label Node (absolute path)
Return value the number of labels. (returns -1 if an error occurs or there is no label.)

Be sure to specify a real index if the label contains an array expression.

Creating a list of label strings in the specified node

```
int getArrayLabels(const std::string label, std::vector<std::string> &labels);
```

label Node (absolute path)
labels List of label strings (output parameters)
Return value the number of labels. (returns -1 if an error occurs.)

Note that this method returns all label strings that match if the label contains an array expression ([@]).

4.3 Usage in a C Program

To use Text Parser library in a C program, include TextParser.h, and invoke the API functions for the C program provided in the TextParser.h.

4.3.1 Include File

When using Text Parser library in a user program, simply include TextParser.h, which provides the API functions for C programs. The data types to be used in the user programs are defined in the TextParserCommon.h.

4.3.2 TextParserValueType

TextParserValueType is the same as that of C++. The definition of TextParserValueType in the TextParserCommon.h is shown in Table 4.1.

4.3.3 TextParserError

TextParserError is the same as that for C++. The definition of TextParserError in the TextParserCommon.h is shown in Tables 4.2 and 4.3.

4.3.4 Creating and Destroying an Instance and Accessing a Singleton

TextParser instances are handled with TP_HANDLE in a C program.

Obtaining the pointer to a TextParser singleton instance

```
TP_HANDLE tp_getInstanceSingleton();
```

Return value the pointer to a TextParser singleton instance

Creating a TextParser instance and obtaining its pointer

```
TP_HANDLE tp_createInstance();
```

Return value the pointer to the TextParser instance

Deleting a TextParser instance

```
int tp_deleteInstance(TP_HANDLE tp_hand);
```

tp_hand Pointer to the TextParser instance to delete
Return value error code (=0: no error), which is defined in `TextParseError`.

4.3.5 File I/O and Memory Release

This subsection describes the following functions: two functions that read a parameter file and store the analysis results of the file in the data structure, and a function that outputs all parameters stored in the data structure.

Reading from a parameter file

```
int tp_read(TP_HANDLE tp_hand,char* file);
```

tp_hand Pointer to a TextParser instance
`file` Input file name
Return value error code (=0: no error), which is defined in `TextParseError`.

Reading from a parameter file (MPI local file version)

```
int tp_read_local(TP_HANDLE tp_hand,char* file);
```

tp_hand Pointer to a TextParser instance
file Input file name
Return value error code (=0: no error), which is defined in `TextParseError`.

Output to a file

```
int tp_write(TP_HANDLE tp_hand,char* file);
```

tp_hand Pointer to a TextParser instance
`file` Output file name
Return value error code (=0: no error), which is defined in `TextParseError`.

The tp_read of the MPI version reads a parameter file in the process of rank = 0. Therefore, be sure to locate the parameter file in a directory that can be accessed by the computer where the rank 0 process will run.
The tp_read_local of the MPI version reads a specified parameter file independently during all processes. The tp_read_local of a non-MPI version does not read a file. It simply returns an error and displays an error message.

The data structure to be written in a file is pre-read from another file, analyzed and processed, and thus the conditional expression of its value with dependency is determined when the file reading is complete.

In addition, the function for discarding the data of the stored parameters can be defined as follows:

Discarding parameter data

```
int tp_remove(TP_HANDLE tp_hand);
```

tp_hand Pointer to a TextParser instance
Return value error code (=0: no error). For further details, see `TextParseError`.

4.3.6 Obtaining a Label and Value (Full Path)

This subsection describes the four functions used to obtain the following: the number of leaves of all parameters stored in the data structure, the label (full path) of the ith leaf specified with index number i, a parameter value by specifying a path, and the type of a parameter value.

Obtaining the number of leaves

```
int tp_getNumberOfLeaves(TP_HANDLE tp_hand,unsigned int* nleaves);
```

tp_hand Pointer to a TextParser instance
`nleaves` The number of leaves
Return value error code (=0: no error), which is defined in `TextParseError`.

Obtaining the ith leaf label

```
int tp_getLabel(TP_HANDLE tp_hand,int i,char* label);
```

tp_hand Pointer to a TextParser instance
i Index number
label Label (full path)
Return value error code (=0: no error), which is defined in TextParseError.

Obtaining a value by specifying a path

```
int tp_getValue(TP_HANDLE tp_hand,char* label,char* value);
```

tp_hand Pointer to a TextParser instance
label Path to the parameter
value Parameter value
Return value error code (=0: no error), which is defined in TextParseError.

Obtaining the value type

```
int tp_getType(TP_HANDLE tp_hand,char* label, int *type);
```

tp_hand Pointer to a TextParser instance
label Path to the parameter
type Type of the parameter value. For further details, see TextParserType in Table 4.1.
Return value error code (=0: no error), which is defined in TextParseError.

4.3.7 Functions for Accessing a Label-Relative Path

This subsection describes the following functions: obtaining the current node, changing the nodes, obtaining the number of child nodes, obtaining the number of leaves, and obtaining a leaf:

Obtaining the current node

```
int tp_currentNode(TP_HANDLE tp_hand,char* node);
```

tp_hand Pointer to a TextParser instance
node Path to the current node
Return value error code (=0: no error). For further details, see TextParseError.

Changing nodes

```
int tp_changeNode(TP_HANDLE tp_hand,char* label);
```

tp_hand Pointer to a TextParser instance
label Label of the destination node (relative path)
Return value error code (=0: no error). For further details, see TextParseError.

Obtaining the number of child nodes of the current node

```
int tp_getNumberOfCNodes(TP_HANDLE tp_hand,int* nnodes);
```

tp_hand Pointer to a TextParser instance
nnodes The number of child nodes
Return value error code (=0: no error). For further details, see TextParseError.

Obtaining the label of a child node of the current node

```
int tp_getIthNode(TP_HANDLE tp_hand,int i,char* label);
int tp_getIthNodeOrder(TP_HANDLE tp_hand,int i,char* label,int order);
```

tp_hand Pointer to a TextParser instance
i Specify the label of the ith node.
label Label of the child node (relative path)
order The output order of the labels is as follows. 0: the same as tp_getIthNode; 1: the index order of an array label; and 2: the order of appearance in a parameter file.
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining the number of leaves in the current node

```
int tp_getNumberOfCLeaves(TP_HANDLE tp_hand,int* nleaves);
```

tp_hand Pointer to a TextParser instance
nleaves The number of leaves in the current node
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining a leaf label of the current node

```
int tp_getIthLeaf(TP_HANDLE tp_hand,int i,char* label);
int tp_getIthLeafOrder(TP_HANDLE tp_hand,int i,char* label,int order);
```

tp_hand Pointer to a TextParser instance
i Specify the label of the ith leaf
label Label of the leaf (relative path)
order Output order of the labels 0: the same as tp_getIthLeaf; 1: the index order of an array label; and 2: the order of appearance in a parameter file
Return value error code (=0: no error). For further details, see `TextParseError`.

4.3.8 Functions for Type Conversion

This subsection describes the functions for converting a parameter value (string) into a specified type:

Conversion of string value type

```
char tp_convertChar(TP_HANDLE tp_hand,char* value, int *error);
short tp_convertShort(TP_HANDLE tp_hand,char* value, int *error);
int tp_convertInt(TP_HANDLE tp_hand,char* value, int *error);
long tp_convertLong(TP_HANDLE tp_hand,char* value, int *error);
long long tp_convertLongLong(TP_HANDLE tp_hand,char* value, int *error);
float tp_convertFloat(TP_HANDLE tp_hand,char* value, int *error);
double tp_convertDouble(TP_HANDLE tp_hand,char* value, int *error);
int tp_convertBool(TP_HANDLE tp_hand,char* value, int *error);
```

tp_hand Pointer to a TextParser instance
value Parameter value (type of charater)
error error code (=0: no error). For further details, see `TextParseError`.
Return value a specified type of value converted from the parameter value.

As for `tp_convertBool`, it will return an int value converted from a string value, as shown in Table 4.6.

Table. 4.6 Conversion of tp_convertBool

String of value	tp_convertBool Return value (int)
true	1
TRUE	1
1	1
false	0
FALSE	0
0	0

4.3.9 Splitting a Vector-Type Value

This subsection describes two functions: obtaining the number of elements in a vector-type parameter and obtaining the ith element (string):

Obtaining the number of elements of a vector-type parameter

```
int tp_getNumberOfElements(TP_HANDLE tp_hand,char* vector_value, int* nvelem);
```

tp_hand Pointer to a TextParser instance
vector_value Value of a vector-type parameter (string)
nvelem The number of elements
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining an element of a vector-type parameter

```
int tp_getIthElement(TP_HANDLE tp_hand,char* vector_value,int ivelem,char* velem);
```

tp_hand Pointer to a TextParser instance
vector_value Value of a vector type parameter (string)
ielem Specify the ielemth element
velem Value of the element (string)
Return value error code (=0: no error). For further details, see `TextParseError`.

4.3.10 Editing Parameter Data

This subsection describes the functions for editing the data of the stored parameters.

Deleting the parameter data

```
int tp_deleteLeaf(TP_HANDLE tp_hand,char* label);
```

tp_hand Pointer to a TextParser instance
label Label of a leaf to delete
Return value error code (=0: no error). For further details, see `TextParseError`.

Updating the parameter data

```
int tp_updateValue(TP_HANDLE tp_hand,char* label,char*value);
```

tp_hand Pointer to a TextParser instance
label Label of a leaf to update
value Leaf value to update. Be sure to set a string that does not change the type of leaf (TP_VALUE_TYPE).
Return value error code (=0: no error). For further details, see `TextParseError`.

Creating the parameter data

```
  int tp_createLeaf(TP_HANDLE tp_hand,char* label,char*value);
```

tp_hand Pointer to a TextParser instance
label Label of a leaf to create
value Value of the leaf to create. If the leaf value is a string, enclose it with double quotation marks.
Return value error code (=0: no error). For further details, see `TextParseError`.

4.3.11 API for Specifying a Range of Parameter Values

Obtaining the parameters by specifying a sequence of numbers with a constant increase or decrease

```
  int tp_splitRange(TP_HANDLE tp_hand,char* value,
     double *from,double *to,double *step);
```

tp_hand Pointer to a TextParser instance
value Value in a string
from Start value
to End value
step Value of a constant increase or decrease
Return value error code (=0: no error). For further details, see `TextParseError`.

Expanding the parameters by specifying a sequence of numbers with a constant increase or decrease

```
int tp_expandRange(TP_HANDLE tp_hand, char* value,
      double* expanded);
```

tp_hand Pointer to a TextParser instance
value Value in a string
expanded Array of the expanded double values
Return value error code (=0: no error). For further details, see `TextParseError`.

Expanding a parameter by specifying an arbitrary sequence of numbers

```
int tp_splitList(TP_HANDLE tp_hand,char* value,double* list,
    int order);
```

tp_hand Pointer to a TextParser instance
value Value in a string
list Array of the expanded double values
order Defined in TextParserSortOrder. TP_SORT_NONE: order of the description; TP_SORT_ASCENDING: ascending order; TP_SORT_DESCENDING: descending order.
Return value error code (=0: no error). For further details, see `TextParseError`.

4.4 Usage in Fortran90

To use Text Parser library in Fortran 90 programs, include TextParser.inc. TextParser.inc exposes API functions for users to access Text Parser library from the user programs. For further details on the error codes of TextParserError and the value types of TextParserValueType used in user programs, see Tables 4.1, 4.2, and 4.3. A parameter string for a function must be initialized with a space, the length of which must be the same as the parameter string itself. For further details, see the examples in the Examples/ directory.

To link the user programs, the option `-lTP_fapi` is required along with the C program options, `-lstdc++, -lTP` and `-L${prefix}/lib`.

4.4.1 Creating and Destroying an Instance, and Accessing a Singleton

TextParser class instances are handled using INTEGER*8 in Fortran programs. Define such instances according to the user programs.

Obtaining the pointer to the TextParser singleton instance

```
integer TP_GET_INSTANCE_SINGLETON(INTEGER*8 ptr)
```

ptr Pointer to the TextParser instance
Return value error code

Creating a TextParser instance and obtaining its pointer

```
integer TP_CREATE_INSTANCE(INTEGER*8 ptr)
```

ptr Pointer to a TextParser instance
Return value error code

Deleting a TextParser instance

```
integer TP_DELETE_INSTANCE(INTEGER*8 ptr)
```

ptr Pointer to a TextParser instance to delete
Return value error code (=0: no error), which is defined in `TextParseError`.

4.4.2 File I/O and Memory Release

This subsection describes the following functions: two functions that read a parameter file and store its analysis result in the data structure, and a function that outputs all parameters stored in this data structure.

Reading from a parameter file

```
INTEGER TP_READ(INTEGER*8 ptr,CHARACTER(len=*) file)
```

ptr Pointer to a TextParser instance
`file` Input file name
Return value error code (=0: no error), which is defined in `TextParseError`.

Reading from a parameter file (MPI local file version)

```
INTEGER TP_READ_LOCAL(INTEGER*8 ptr,CHARACTER(len=*) file)
```

ptr Pointer to a TextParser instance
file Input file name
Return value error code (=0: no error), which is defined in `TextParseError`.

Output to a file

```
INTEGER TP_WRITE(INTEGER*8 ptr, CHARACTER(len=*) file)
```

ptr Pointer to a TextParser instance
`file` Output file name
Return value error code (=0: no error), which is defined in `TextParseError`.

TP_READ of the MPI version reads a parameter file in the process of rank = 0. Therefore, be sure to locate the parameter file in a directory that can be accessed by a computer where a rank = 0 process will run.

TP_READ_LOCAL of the MPI version reads a specified parameter file independently during all processes. TP_READ_LOCAL of a non-MPI version does not read a file. It simply returns an error and displays an error message. Be sure NOT to use the READ_LOCAL for non-MPI programs.

The data structure to be written in a file is pre-read from another file, analyzed and processed, and thus the conditional expression of its value with dependency is determined when the file reading is complete.

In addition, the function of discarding the data of the stored parameters can be defined as follows:

Discarding the parameter data

```
INTEGER TP_REMOVE(INTEGER*8 ptr);
```

ptr Pointer to a TextParser instance
Return value error code (=0: no error). For further details, see `TextParseError`.

4.4.3 Obtaining a Label and Value (Full Path)

This subsection describes the four functions used to obtain the following: the number of leaves of all parameters stored in the data structure, the label (full path) of the ith leaf specified with index number: i, the parameter value by specifying a path, and the type of a parameter value.

Obtaining the number of all leaves

```
INTEGER TP_GET_NUMBER_OF_LEAVES(INTEGER*8 ptr,INTEGER nleaves)
```

ptr Pointer to a TextParser instance
`nleaves` The number of leaves
Return value error code (=0: no error), which is defined in `TextParseError`.

Obtaining the ith leaf label

```
integer TP_GET_LABEL(INTEGER*8 ptr,INTEGER i,CHARACTER(len=*) label)
```

ptr Pointer to a TextParser instance
`i` Index number. Indexes of Fortran starts with one.
`label` Label (full path)
Return value error code (=0: no error), which is defined in `TextParseError`.

Obtaining a value by specifying a path

```
INTEGER TP_GET_VALUE(INTEGER*8 ptr,CHARACTER(len=*) label,CHARACTER(len=*) value)
```

ptr Pointer to a TextParser instance
`label` Path to the parameter (string)
`value` Parameter value (string)
Return value error code (=0: no error), which is defined in `TextParseError`.

Obtaining the value type

```
INTEGER TP_GET_TYPE(INTEGER*8 ptr,CHARACTER(len=*) label,INTEGER type)
```

ptr Pointer to a TextParser instance
`label` Path to the parameter
`type` Type of the parameter value. For further details, see `TextParserType` in Table 4.1.
Return value error code (=0: no error), which is defined in `TextParseError`.

4.4.4 Functions for Accessing a Label-Relative Path

This subsection describes the following functions: obtaining the current node, changing nodes, obtaining the number of child nodes, obtaining the number of leaves, and obtaining a leaf.

Obtaining the current node

```
INTEGER TP_CURRENT_NODE(INTEGER*8 ptr,CHARACTER(len=*) node)
```

ptr Pointer to a TextParser instance
node Path to the current node
Return value error code (=0: no error). For further details, see `TextParseError`.

Changing nodes

```
INTEGER TP_CHANGE_NODE(INTEGER*8 ptr,CHARACTER(len=*) label);
```

ptr Pointer to a TextParser instance
label Label of the destination node (relative path)
Return value error code (=0: no error). For further details, `TextParseError`.

Obtaining the number of child nodes of the current node

```
INTEGER TP_GET_NUMBER_OF_CNODES(INTEGER*8 ptr,INTEGER nnodes)
```

ptr Pointer to a TextParser instance
nnodes The number of child nodes
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining a label of a child node of the current node

```
INTEGER TP_GET_ITH_NODE(INTEGER*8 ptr,INTEGER i,CHARACTER(len=*) label)
INTEGER TP_GET_ITH_NODE_ORDER(INTEGER*8 ptr,INTEGER i,CHARACTER(len=*) label,
                              INTEGER order)
```

ptr Pointer to a TextParser instance
i Specify the label of the ith node.
label Label of the child node (relative path)
order output order of labels 0: the same as TP_GET_ITH_NODE; 1: the index order of an array label; and 2: the order of appearance in a parameter file
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining the number of leaves in the current node

```
INTEGER TP_GET_NUMBER_OF_CLEAVES(INTEGER*8 ptr,INTEGER* nleaves);
```

ptr Pointer to a TextParser instance
nleaves The number of leaves in the current node
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining a leaf label of the current node

```
INTEGER TP_GET_ITH_LEAF(INTEGER*8 ptr,INTEGER i,CHARACTER(len=*) label)
INTEGER TP_GET_ITH_LEAF_ORDER(INTEGER*8 ptr,INTEGER i,CHARACTER(len=*) label,
                integer order)
```

ptr Pointer to a TextParser instance
i Specify the label of the ith node.
label Label of the leaf (relative path)
order Output order of labels 0: the same as TP_GET_ITH_LEAF; 1: the index order of an array label; and 2: the order of appearance in a parameter file
Return value error code (=0: no error). For further details, see `TextParseError`.

4.4.5 Functions for Type Conversion

This subsection describes the functions for converting a parameter value (string) into a specified type.

Conversion of string value type

```
INTEGER*1 TP_CONVERT_CHAR(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*1 TP_CONVERT_INT1(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*2 TP_CONVERT_SHORT(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*2 TP_CONVERT_INT2(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*4 TP_CONVERT_INT(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*4 TP_CONVERT_INT4(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
INTEGER*8 tp_CONVERT_INT8(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
REAL TP_CONVERT_FLOAT(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
REAL*8 TP_CONVERT_DOUBLE(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
LOGICAL TP_CONVERT_LOGICAL(INTEGER*8 ptr,CHARACTER(len=*) value, INTEGER error)
```

ptr Pointer to a TextParser instance
value Parameter value (type of character)
error error code (=0: no error). For further details, see `TextParseError`.
Return value A value of the type specified by the user.

Table. 4.7 Conversion of tp_CONVERT_

String of value	TP_CONVERT_LOGICAL Return value (LOGICAL)
true	.true.
TRUE	.true.
1	.true.
false	.false.
FALSE	.false.
0	.false.

4.4.6 Splitting a Vector-Type Value

This subsection describes two functions: obtaining the number of elements in a vector-type parameter, and obtaining the ith element (string):

Obtaining the number of elements of a vector-type parameter

```
INTEGER TP_GET_NUMBER_OF_ELEMENTS(INTEGER*8 ptr,CHARACTER(len=*) vector_value,
                                  INTEGER nvelem)
```

ptr Pointer to a TextParser instance
vector_value Value of a vector-type parameter (string)
nvelem The number of elements
Return value error code (=0: no error). For further details, see `TextParseError`.

Obtaining an element of a vector-type parameter

```
INTEGER tp_get_ith_element(INTEGER*8 ptr,CHARACTER(len=*) vector_value,
                           INTEGER ivelem,CHARACTER(len=*) velem)
```

ptr Pointer to a TextParser instance
vector_value Value of a vector-type parameter (string)
ielem Specify the ielemth element
velem Value of the element (string)
Return value error code (=0: no error). For further details, see `TextParseError`.

4.4.7 Editing Parameter Data

This subsection describes the functions for editing the data of the stored parameters:

Deleting the parameter data

```
integer TP_DELETE_LEAF(INTEGER*8 ptr,CHARACTER(len=*) label)
```

ptr Pointer to a TextParser instance
label Label of a leaf to delete
Return value error code (=0: no error). For further details, see `TextParseError`.

Updating the parameter data

```
integer TP_UPDATE_VALUE(INTEGER*8 ptr,CHARACTER(len=*) label,
                        CHARACTER(len=*) value)
```

ptr Pointer to a TextParser instance
label Label of a leaf to update
value Leaf value to update. Be sure to set a string that does not change the type of the leaf (TP_VALUE_TYPE).
Return value error code (=0: no error). For details, see `TextParseError`.

Creating the parameter data

```
integer TP_CREATE_LEAF(INTEGER*8 ptr,CHARACTER(len=*) label,
                       CHARACTER(len=*) value)
```

ptr Pointer to a TextParer instance
label Label of a leaf to create
value Value of the leaf to create. If the value is a string, enclose it with double quotation marks.
Return value error code (=0: no error). For details, see `TextParseError`.

4.4.8 API for Specifying a Range of Parameter Values

Obtaining the parameters by specifying a sequence of numbers with a constant increase or decrease

```
  INTEGER TP_SPLIT_RANGE(INTEGER*8  ptr,CHARACTER(len=*) value,
     REAL*8 from,REAL*8 to,REAL*8 step);
```

ptr Pointer to a TextParser instance
value Value in a string
from Start value
to End value
step Value with a constant increase or decrease
Return value error code (=0: no error). For details, see `TextParseError`.

Pointer to a TextParser instance

```
INTEGER TP_EXPAND_RANGE(INTEGER*8 ptr, CHARACTER(len=*) value,
      REAL*8* expanded(*));
```

ptr Pointer to a TextParser instance
value Value in a string
expanded Array of an expanded numerical value
Return value error code (=0: no error). For further details, see `TextParseError`.

Expanding a parameter by specifying an arbitrary sequence of numbers

```
INTEGER TP_SPLIT_LIST(INTEGER*8  ptr,CHARACTER(len=*) value,REAL*8 list(*),
    INTEGER order);
```

ptr Pointer to a TextParser instance
value Value in a string
list Array of an expanded numerical value
order Defined in TextParserSortOrder. 0: order of descriptions; 1: ascending order; and 2: descending order
Return value error code (=0: no error). For further details, see `TextParseError`.

Chapter 5

Parameter Description

The parameter database of the Text Parser library stores a leaf, which is a pair made up of a label and a value, in the node hierarchy. The parameter database consists mainly of labels, nodes, leaves, and values.

5.1 Label

A label is a discriminator of a node or leaf. The specifications of a label are as follows:

- A discriminator of a node and leaf

```
[Node]
  label {    }
[Leaf]
  label = value
```

- A string on the left side of the following symbols: "=","{", "==" in @dep, and "!=" in @dep
  - A string without a space. However, it does not matter whether a new line or space exists between a string and a symbol.
  - A string enclosed with double quotation marks is equal to a string without double quotation marks.
  - Available characters are [a-zA-Z0-9_-].
  - A label can be written in a node-path like style. (hereafter, called a label path.) Both an absolute path and a relative path are available. [/.] is also available.
  - A string comparison is case-insensitive.

  See the following example:

```
[Normal]
 The following four notations indicate the same labels:
 foo, "foo", FOO, and "FOO".
[Absolute path]
 /foo/baz or "/foo/baz"
[Relative path]
 ./foo/baz or "./foo/baz"
 foo/baz or "foo/baz"
 ../bar or "../bar"
```

- Nodes in the same hierarchy cannot have the same label. In addition, leaves in the same node hierarchy also cannot have the same label.

```
Possible example
foo {
    qux { baz = "val1" }
    qux { x = "val2" }
}
```

```
Error example
foo {
    qux = 1
    qux = 2   // error
}
```

- A node and a leaf cannot have the same label in a single node hierarchy.

```
foo {
    qux { baz = "val1"}
    qux = "val2"   // error
}
```

- An absolute path cannot have more than one of the same labels.

```
qux {
    qux {         // error
         baz = "val1"
    }
}
qux {
    qux = "val1"   // error
}
```

- A label is defined in an array shape by describing "[@]" at the end of a label string. When an array shape label is parsed, @ is converted into a unique suffix initialized in the node.

```
 foo {
     bar[@] { param[@]=1,
              param[@]=2
     }
     bar[@] { param[@]=3,
              param[@]=4
     }
}
After being parsed, it is converted as follows:
foo {
    bar[0] { param[0]=1,
             param[1]=2
    }
    bar[1] { param[0]=3,
             param[1]=4
    }
}
```

- In a single node, the same array shaped label cannot be defined for a node or a leaf.

```
foo {
    param[@] { abc = 1 }
    param[@]=1    // error
}
```

- An absolute path cannot have more than one of the same array labels.

```
param[@] {
        param[@] {
                param[@] = 1
        }
}
```

- Labels do not need to be indented; the following two examples have the same meaning.

```
Example 1
one {
   two  {
                three = 1
          }
}

Example 2
one {
two{
three = 1
}
}
```

5.2 Node

A node is identified by its name and location, as defined by its label. A node retains a leaf or another node. The specifications of a node are as follows:

- A node retains a leaf or another node. A node starts with a label; the elements of the node start with ”{”, and end with ”}”. Nodes can also be nested. The following three examples have the same meaning.

```
label { }
(Example)
foo {
}
foo  {    }
foo
{
}
```

- A node can be defined using a relative path or an absolute path. A relative node path is available with “../” which means a node above and with “./” which means the current node.

```
Absolute path
/foo/qux {
}

Relative path: The following two examples have the same meaning.

Example 1
foo {
    ./qux/abc  {  //  ./ is omittable.
       ../cde/ {    }
    }
}

Example 2
foo {
    ./qux/abc { }
    ./qux/cde { }
}
```

- One node can be divided to define

```
foo {
    baz=1
}
foo {
    abc=2
}
```

- A node hierarchy is placed between curly brackets; curly brackets do not need to be indented, and it does not matter whether a new line exists or not. The following two examples have the same meaning.

```
Example 1
one { two{ three = 1  }
  }

Example 2
one        {
 two{      three = 1
   }
        }
```

5.3 Leaf

A pair made up of a label and the value of a node element is called a “leaf.” The specifications of a leaf are as follows:

- A leaf is a node that connects a label to a value with an assignment operator. Multiple leaves are separated using a comma, space, or a new line.

```
Label = value
qux="val"
param0=1, param1=2
param0=1 param1=2
```

- Defining a leaf using an absolute path and a relative path

```
Absolute path
/foo/qux="val1"
Relative path
./bar/baz=1   //  ./ can be omitted.
```

- A leaf can be described in any order within a parameter file.

5.4 Value

A leaf is the result of a pairing of a label and a leaf, and has one of five types of values, the specifications of which are as follows:

1. String
   - ASCII strings enclosed by double quotation marks.
   - Available characters are [a-zA-Z0-9_-].
   - A string comparison is case-insensitive.

   ```
   Example of a string
   "ON" "on"
   ```

2. Numerical value
   - integer
     - A space can exist after "(+/-) 0-9" in "+/-"
     - A spare 0 can also exist

     ```
     Example of integer
     123
     0
     + 123
     - 0
     01
     ```
   - floating-point number
     - A space can exist after "+/-" in "(+/-) 0-9.(0-9)" or "(+/-) .0-9 ".

     ```
     Example of floating-point number
     0.10
     + 0.2
     - 0.3
     3.
     .123
     ```
   - Exponent
     - A space can exist after "+/-" in "(+/-) 0-9.(0-9) e/d (+/-) 0-9" and "(+/-).0-9 D/E (+/-) 0-9".
     - A space can also exist before or after e/d.

     ```
     Example of exponent
     1.0 e 10
     + 0.1 E - 5
     - 1.2 d 3
     .3 e - 4
     ```
3. Vector
   - A set of ordered strings or ordered numerical values
   - Starts with a left parenthesis "(" and ends with a right parenthesis ")."
   - Values are discriminated using a comma.
   - All value elements in a vector must be of the same type.
   - A vector of an undefined value is available (UNDEF).
   - Vectors cannot be nested.

   ```
   Example of a vector
   (1.0 e 10, 1.0 e 10 , 1.0 e 10 )
   (+ 0.1 E - 5 ,- 0.3 E - 5 )
   (- 1.2 d 3,- 1.2, 3 ,.3 e - 4)
   ("ONE","TWO","THREE")
   ( UNDEF, UNDEF, UNDEF )
   ```

4. Numerical Limits

- Values of numerical limits in limits.h are available. Specify the values in the upper characters as follows:
  ```
  CHAR_MIN //char the minimum value
  CHAR_MAX  //char the maximum value
  SHORT_MIN //short the minimum value
  SHORT_MAX  //short the maximum value
  INT_MIN //int the minimum value
  INT_MAX  //int the maximum value
  LONG_MIN //long the minimum value
  LONG_MAX  //long the maximum value
  LONGLONG_MIN //longlong the minimum value
  LONGLONG_MAX  //longlong the maximum value
  FLOAT_MIN //float the minimum value
  FLOAT_MAX  //float the maximum value
  DOUBLE_MIN //double the minimum value
  DOUBLE_MAX  //double the maximum value
  ```

5. A value with dependency
   - This value can be controlled based on a C-like ternary operator expression specified after "@dep."
   - Write a conditional expression in a location that starts with a left parenthesis and ends with a right parenthesis after "@dep."
   - The left term of a conditional expression is a label path of a dependent parameter, and the right term is the possible value of the dependent parameter.
   - A relative path is available for a label path.
   - An array-shaped label, e.g., ”param[@]” cannot be used as a label path.
   - The value of the right term of a conditional expression must be a string or a numerical value.
   - A vector, an undefined value, or a value with dependency cannot be specified as a value of the right term of a conditional expression.
   - "==" and "!=" are available as a conditional operators.
   - A conditional expression is connected with the operators "&&" or "||" and is processed in the order shown in the parenthesis. Multiple specifications are available.
   - When a value of the right term of a conditional expression is a numerical value, the value will be converted into a double, and whether "==" or "!=" is to be used is determined.
   - When a conditional expression is true, a value is written in a location next to "?" after the conditional expression. When a conditional expression is false, a value is written in a location next to ":."
   - A vector is available. The dimensions of a vector to the left of and the right of ":" may be different.
   - The value type to the left of and the right of ":" may be different.
   - An undefined value is available; however, a value with dependency cannot be specified as a value.
   - If a label for a value with dependency is not defined, the evaluation will be executed again after all parameters are parsed.
   - Sets an undefined value, UNDEF, to a value with dependency that finally references an undefined label, and displays a warning.

```
Description of a value with dependency
@dep ( conditional expression ) ? value : value
Conditional expression
    label == value
    label != value
    conditional expression && conditional expression
    conditional expression || conditional expression

Example of a value with a value with dependency
@dep (swt == "on") ? 1 : 2
@dep ((swt == "on")) ? 1 : 2
@dep ( a == 1) ? (1,2) : (0,1,2)
@dep ( a == 1) ? (1,2) : ("a","b")
@dep ( a == 1) ? 5 : UNDEF
@dep ( (swt1==0) && (swt2==1) ) ? "a" : "b"
```

```
@dep ( (swt1==0) || (swt2!=1) ) ? "a" : "b"
```

6. Undefined value
   - To clearly indicate that a value is undefined, UNDEF can be used.
   - A warning will be shown for an undefined value while parsing a parameter. If an undefined value is obtained as a parameter, a warning will be displayed.

```
Example of an undefined value
baz=UNDEF
```

7. Specifying a range of values using an arbitrary sequence of numbers
   - A range of values can be specified by describing an arbitrary sequence of numbers: @list=(val1,val2,val3,...valN). The numerical values of @list should follow the description rule for the numerical values of TextParser.
     ```
     Example of specifying a value range using an arbitrary sequence of numbers:
     label = @list(1,3,5,6.0,3.14e-03,-7)
     ```
8. Specifying a range of values using a sequence of numbers with a constant increase or decrease
   - The description format is @range(from,to,step). From,to,step are the start value, end value, and value of a constant increase or decrease, respectively. The numerical values of @range should follow the description rule for the numerical values of TextParser. An error will occur if the increase and/or decrease direction of the start value and the end values is inconsistent with the sign of the value of the constant increase or decrease.
     ```
     Specifying a range of values using a sequence of numbers
                                with a constant increase or decrease:
     label = @range(1,10,2) //is the same value as @list(1,3,5,7,9).
     label = @range(1,3) //if step is omitted, it means 1 or -1.
                                 This is the same value as @list(1,2,3).
     ```

5.5 New Line

An element can start on a new line in a parameter description. However, labels, string values, numerical values, undefined values (UNDEF), or "`@dep`" of the dependence cannot contain a new line.

5.6 Comment

C/C++ like /**/ and // are available.

5.7 Examples of Parameter Files

The Examples/ directory contains many examples of parameter parser files, which are stored in the Examples/tpp_examples directory. They consist of the following files:

- Syntactically correct examples
  correct_basic_*.txt for testing a basic tree structure and values (numerical value and numerical vector).
  correct_string_*.txt for testing a value (string or string vector).
  correct_label_*.txt for testing label specifications (node/leaf).
  correct_labelarray_*.txt for testing an array label specification (node/leaf).
  correct_cond_*.txt for testing a value with dependency: `''@dep.''`
  correct_range_list_*.txt for testing `''@range''` and `''@list.''`
- Syntactically incorrect examples
  incorrect_basic_*.txt for testing a basic tree structure and values (numerical value and numerical vector).
  incorrect_label_*.txt for testing label specifications (node/leaf).
  incorrect_labelarray_*.txt for testing an array label specification (node/leaf).
  incorrect_cond_*.txt for testing a value with dependency: `''@dep.''`
  correct_range_*.txt for testing `''@range.''`
  correct_list_*.txt for testing `''@range''` and `''@list.''`

Chapter 6

Update Information

This chapter provides information on updates of Text Parser library.

6.1 Update Information

- 2014-06-26
  - Initial Release of English version