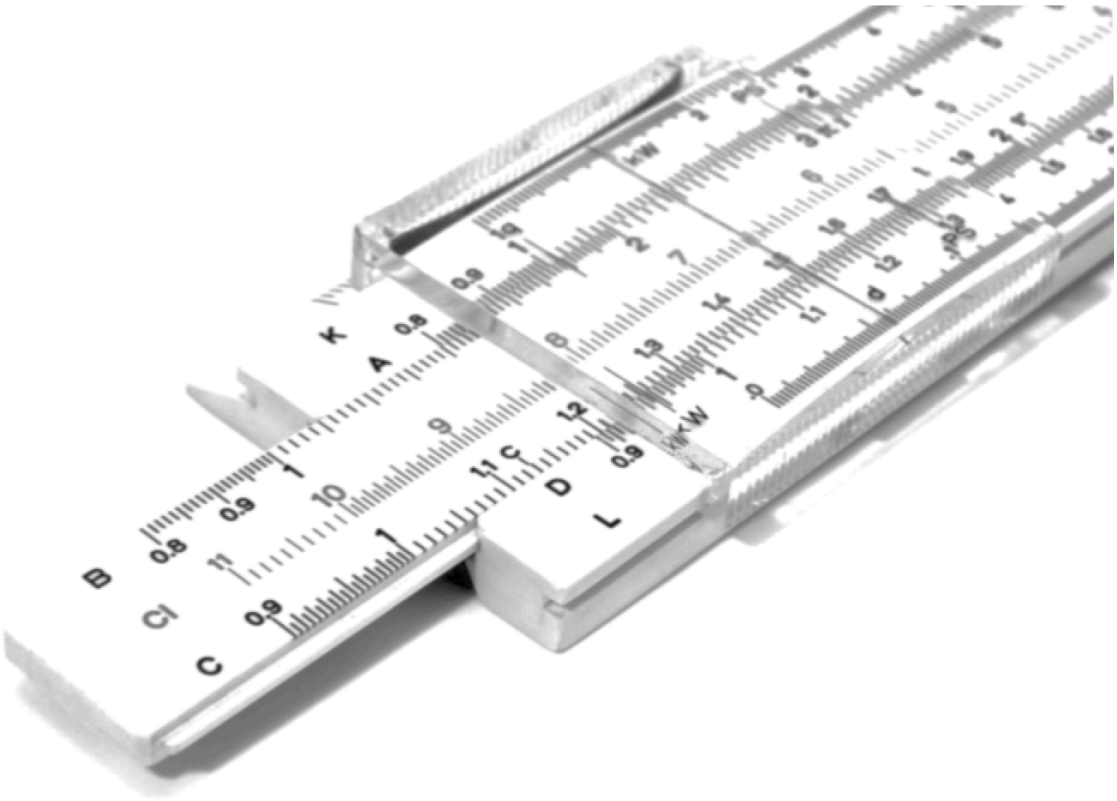
Library Guide

Version 1.0
January 2016



CryoSoft

VARIABLES

Parametric Input Extension Library

DISCLAIMER

Even though CryoSoft has carefully reviewed this manual, CRYOSOFT MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS PROVIDED "AS IS", AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

Contents

Roadmap

Before you start

This manual is the reference user's guide for the *Variables* Library, a FORTRAN extension that improves customization and simplifies programming of External Routines in the CryoSoft simulation codes. Throughout this manual we assume that the reader is familiar with the installation of the CryoSoft codes [1], the specific suite of simulation codes that make use of External Routines, and has a basis of FORTRAN programming in UNIX/Linux environment.

How to use this manual

This manual is structured as follows:

- Chapter 1 contains a brief and general introduction to the functions and structure of the *Variables* Library.
- Chapter 2 contains information on the syntax of the input block where the user Variables are defined and the values assigned.
- Chapter 3 describes the calling protocol of the functions that give access to the user's defined Variables.
- Chapter 4 deals with troubleshooting and error messages;
- Chapter 5 gives the references and a general bibliography for documentation.

Beginners to the *Variables* Library should read chapters 1, 2 and 3 in sequence. Experienced users will use chapter 3 as a reminder for programming. Chapter 4 is designed as an indexed glossary for error messages and associated actions.

CHAPTER 1

Introduction

What is the *Variables* Library

The *Variables* Library is an extension available in most CryoSoft simulation codes (e.g. THEA, FLOWER, POWER, HEATER, and other codes in future), used to enlarge the set of standard input keywords by an arbitrary number of named and typed variables. This takes place in two steps. Firstly, the new variables are defined according to a simple syntax in the parsed input file of a simulation code. Once parsed from input, the new variables can then be accessed (retrieved and modified) from the *External Routines* of the code, using simple FORTRAN calls.

In essence, the *Variables* Library provides a direct communication channel between input and the External Routines, which is useful to avoid direct coding of parameters, avoid the need for re-compilation of the simulation code with its *External Routines*, helps in making inputs more clear, improves the traceability of the simulation cases, and can be used for *scripting* (automatic input generation and parametric runs).

Variables parsing

The variables to be addressed by the *Variables* Library are defined as a part of the input of the CryoSoft simulation code. The user will do this by appending a `Variables` block to the standard input of the code. The syntax of the `Variables` block resembles closely the syntax used in the rest of the input blocks, making the interpretation of the commands and bookkeeping straightforward. The syntax of the `Variables` block is defined in the User's Manuals of the CryoSoft simulation codes, and reported for convenience in Chapter 2.

Library description

The *Variables* Library is the set of FORTRAN routines, used to define, store and retrieve named and typed variables, with names *VariableName* that can be any string up to 256 characters.

The Library supports the following variable types:

- Integer, a 32 bit signed integer number;
- Real, a 32 bit signed, single precision real number with 7 significant digits;
- Character, a string of up to 256 ASCII characters.

The Library provides the following basic functionalities on the variables:

- Variable initialization, to reset all definitions and empties the memory buffer. This function is necessary at the start of a simulation program (must be the first call to the library), hence we document it later, but it is not necessary to the end user's of the CrySoft simulation codes as it is called by the simulation code itself to prepare the environment;
- Variable definition, to define a new variable *VariableName* of a given type, and set the initial value of the variable to zero
- Variable check, to verify the existence of a variable *VariableName*;
- Variable assignment, to assign a value to a variable *VariableName* defined earlier;
- Variable retrieval, to retrieve the value of a variable *VariableName* defined earlier.

The routine calling protocol for the *Variables* Library is defined in Chapter 3.

System requirements

The *Variables* Library is a part of the standard CryoSoft codes installation. At the time when this manual is written, the platforms where the *Variables* Library has been developed is:

- Macintosh running MacOS-X (10.8.5 and higher) under XQuartz,(2.7.4) gcc (4.8.1) with gfortran.

At different time of the development and production, the library has been installed and tested on the following platforms:

- Mac-OS X (10.2 and higher) operating system;
- GNU/Linux operating system (most distributions).
- INTEL PC's running RedHat Linux OS;
- IBM-RISC workstations running the AIX-V4 operating system and later;
- SUN-SPARC workstation running the Solaris OS operating system;
- DEC-ALPHA workstation running the OSF-1 operating system;
- HP workstations running HP-UX OS;
- Windows-2000 and Windows-XP operating system, with an installed CYGWIN environment (the reference version tested is CYGWIN 1.5.24-2).

The Library obeys FORTRAN 77 standards, as supported by most compilers. Porting and using in different architectures should hence be straightforward. Contact us for advice and tests on a specific installation.

CHAPTER 2

Variables Input Reference

Structure and syntax

Variables are read by the input interpreter that parses and analyzes the syntax and the grammar of the various entries in an input file for one of the CryoSoft simulation codes. In general, the input file contains a series of blocks, as detailed in the specific manual of the simulation codes. The block defining variables has the following general structure:

```
Begin Variables
        VariableType     VariableName     value
        VariableType     VariableName     value
        ……..

        VariableType     VariableName     value
End
```

where:

- *VariableType* is the type of variable, to be chosen among the set of keywords described in the following sections,
- *VariableName* is the name assigned to the specific variable, that can be any string up to 256 characters, and
- value is the alphanumerical value assigned.

The structure and content of the block must comply with the following rules and conventions:

- the identifier of a variable and the corresponding value(s) can appear at any position on the line, they can carry on to several lines and must be separated by blanks or tabs;
- the interpretation is case insensitive;
- abbreviations of the keys are not allowed;
- a character ';' in any position of the command line indicates that the remainder of the line must be considered as a comment. If the ';' is the first character in a line, then the whole line is ignored;
- repeated variable assignation overrides previous values and is not checked at read-in time;

Input reference

The following table contains, in alphabetical order, the keywords defining the variable types, and the associated convention for the variable value. Predefined values are reported in `Courier`.

VariableType	Meaning
`Character`	*VariableName* is a string, whose Value is read as a text, delimited by apexes if the text contains a blank (not recommended)
`Integer`	*VariableName* is an integer, whose Value is read according to FORTAN READ conventions
`Real`	*VariableName* is a real, whose Value is read according to FORTAN READ conventions (floating point or scientific notation)

CHAPTER 3

Variables Library Functions

Warning The use of the Variables Library, within or independent of the External Routines, give unlimited access to the data structure used by the main program being customized. Improper programming of Variables and External Routines can therefore corrupt operation and lead to evident or concealed malfunctions and generate manifest or hidden errors in the computed results. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Linking the *Variables* Library

The functions of the *Variables* Library that give access to the parsed variables are FORTRAN functions. These functions are part of the standard libraries linked by the CryoSoft simulation codes. No code modification, or adaptation is required to have access to these functions.

Calling protocol

The following sections describe the calling protocol for the *Variables* Library functions.

Note In the tables below we use the following convention for the type of variables:

C	character (a string delimited by blanks, tabs or apices)
R	real (a number in floating point or engineering notation)
I	integer (an integer number)

Typing must be respected to avoid memory mis-match.

Note FORTRAN unit numbers above 50 are reserved by the CryoSoft library for internal use, and should not be allocated for read/write operations. Any allocation or use of units above 50 can result in I/O errors or malfunctions.

Variables Initialization

```
subroutine InitializeVariables
```

Resets all *Variables* definitions, empties memory buffer. This is the first call to the *Variables* Library, and is done by the CryoSoft simulation codes. The user has no need to call this routine for a normal usage of the *Variables* Library.

Variable Check

```
logical function checkVariable(Name)
```

Check whether a Variable with a given Name has already been defined. Returns logical `.true.` if this is the case, `.false.` in case the variable is not defined.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the variable to be checked (string of 256 characters at most)

Variable Definition

```
logical function newVariable(Name,Type)
```

Define a new variable with a given Name and of a given Type. Returns logical `.true.` if the definition has succeeded, `.false.` in case the variable cannot be defined, e.g. because it is already defined.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the new variable to be defined, string of up to a maximum of 256 characters
`Type`	C	(-)	Variable type, character with the following possible values: `C` character, a string of up to a maximum of 256 characters, `R` real, `I` integer

Variable Assignment

```
logical function putIVariable(Name,Value)
```

Assign an integer Value to a variable Name of type integer. Note that the assignment is typed, and both variable type and Value have to be the same. Returns logical `.true.` if the assignment has succeeded, `.false.` in case the variable cannot be given the Value, e.g. because it is not yet defined, or because of a mis-match of type.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be assigned, string of up to a maximum of 256 characters, the variable must be of type integer
`Value`	I	(-)	Value to be assigned to the Variable, integer

`logical function` **`putRVariable`**`(Name,Value)`

Assign a real Value to a variable Name of type real. Note that the assignment is typed, and both variable type and Value have to be the same. Returns logical `.true.` if the assignment has succeeded, `.false.` in case the variable cannot be given the Value, e.g. because it is not yet defined, or because of a mis-match of type.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be assigned, string of up to a maximum of 256 characters, the variable must be of type real
`Value`	R	(-)	Value to be assigned to the Variable, real

`logical function` **`putCVariable`**`(Name,Value)`

Assign a character Value to a variable Name of type character. Note that the assignment is typed, and both variable type and Value have to be the same. Returns logical `.true.` if the assignment has succeeded, `.false.` in case the variable cannot be given the Value, e.g. because it is not yet defined, or because of a mis-match of type.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be assigned, string of up to a maximum of 256 characters, the variable must be of type character
`Value`	C	(-)	Value to be assigned to the Variable, string of up to a maximum of 256 characters

Variable Retrieval

`logical function` **`getIVariable`**`(Name,Value)`

Retrieve the integer Value of a variable Name of type integer. Note that the assignment is typed, and both variable type and Value have to be the same. Returns logical `.true.` if the retrieval has succeeded, `.false.` in case the variable Value cannot be retrieved, e.g. because it is not yet defined, or because of a mis-match of type.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be retrieved, string of up to a maximum of 256 characters, the variable must be of type integer
`Value`	I	(-)	Value of the Variable as retrieved, integer

`logical function` **`getRVariable`**`(Name,Value)`

Retrieve the real Value of a variable Name of type real. Note that the assignment is typed, and both variable type and Value have to be the same. Returns logical `.true.` if the retrieval has succeeded, `.false.` in case the variable Value cannot be retrieved, e.g. because it is not yet defined, or because of a mis-match of type.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be retrieved, string of up to a maximum of 256 characters, the variable must be of type real
`Value`	R	(-)	Value of the Variable as retrieved, real

`logical function` **`getCVariable`**`(Name,Value)`

Retrieve the character Value of a variable Name of type character. Note that the assignment is typed, and both variable type and Value have to be the same. Returns logical `.true.` if the retrieval has succeeded, `.false.` in case the variable Value cannot be retrieved, e.g. because it is not yet defined, or because of a mis-match of type.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be retrieved, string of up to a maximum of 256 characters, the variable must be of type character
`Value`	C	(-)	Value of the Variable as retrieved, string of up to a maximum of 256 characters

`logical function` **`getVariableType`**`(Name,Type)`

Retrieve the Type of a variable Name. Returns logical `.true.` if the retrieval has succeeded, `.false.` in case the variable Type cannot be retrieved, e.g. because the variable is not yet defined.

Parameter	Type	Units	Meaning
`Name`	C	(-)	Name of the existing variable to be retrieved, string of up to a maximum of 256 characters
`Type`	C	(-)	type of the Variable as retrieved, character with the following possible values: `C` character, a string of up to a maximum of 256 characters, `R` real, `I` integer

CHAPTER 7

Troubleshooting and Errors

The *Variables* Library does not generate error messages, but can return error conditions in when defining and using Variables. The error conditions are then signaled by the calling CryoSoft simulation code. Error messages from the CryoSoft simulation codes are reported to the output ASCII log file and to the standard output. The form of a typical error report is the following

```
ERROR in procedure <procedure name>: <error message>
called by <calling procure> at position <n>
called by <calling procure> at position <m>
......
```

where <*procedure name*> is the name of the routine where the error occurred and <*error message*> reports a short description of the error situation. This line is followed by the trace of the <*calling procedure*> up to the main program. In case of queries about error conditions, please take care to report error messages completely, including the calling trace.

Errors can be generated at four different levels in the code:

- input parsing and syntax errors;
- data consistency errors;
- runtime errors;
- internal consistency errors.

Input parsing errors

Input parsing and syntax errors are detected during the interpretation of the input file. They indicate that the variable naming, the command syntax or the type and number of numerical data in the input file are incorrect. Verify syntax in the input file in this case.

Data consistency errors

Data consistency errors are detected when input data are not coherent among themselves and would result in a model that cannot be analyzed. Typical case is an attempt to define the same variable in multiple instances. Verify the consistency of the input data in this case.

Runtime errors

Runtime errors from the *Variables* Library are detected when the number of variables exceeds the maximum allowed. The maximum number of variables is determined by the requested memory allocation in the FORTRAN include file:

```
~/CryoSoft/src/library/miscellanea/variables.inc
```

where a number of parameters are set statically. The parameter affecting memory allocation is the following, with the associated meaning:

Parameter	Meaning
`MaxVariables`	maximum number of variables

The version of the code you received can be modified by adjusting this parameters as desired. The code then needs to be compiled and link-edited as explained in the installation manual you received [1].

Warning Modification of dimensioning parameters affects memory allocation. Improper programming of parameters can therefore corrupt operation and lead to evident or concealed malfunctions and generate manifest or hidden errors in the computed results. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Internal consistency errors

Internal consistency errors indicate corruption of the internal data structure of a program. The *Variables* Library does not detect and signal internal consistency errors. However, improper use of the Library in conjunction with customized External Routines can cause an internal consistency error in a CryoSoft simulation code. This type of error diagnoses a severe fault. If you are using the *Variables* Library and External Routines, verify their consistency with the calling protocol. Report internal consistency errors to us.

CHAPTER 8

References

[1] CryoSoft Installation Manual, Version 8.1, 2016.