ConfigurationDesk

Syntax of the TRC File

For ConfigurationDesk 6.7

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Document

Objective

If you use custom code, the variables in it are initially not accessible to the experiment software. To make the global variables accessible, you must provide an additional variable description file. When writing a user variable description file, you must use the syntax described in this document, and name it `<model>_usr.trc`. During the build process, the user file is inserted into the main variable description file. It must be created before the build process is started. It has to be located in the working folder of the behavior model. You can write variable description files also for each referenced model in your behavior model.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
?	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
📖	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.
To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the 📄 icon in dSPACE Help. The PDF opens on the first page.

Syntax of the TRC File

Introduction

The TRC file provides information on the variables of a real-time application that is required to connect variables to instruments in a ControlDesk layout, for example. It is an ASCII file that can either be generated automatically by ConfigurationDesk, or written manually.

> **Note**
>
> If you write a TRC file manually, you must adhere to the syntax of the TRC file. Then, you can easily switch from a simulation on the Simulink platform to an application running on a dSPACE real-time board.

TRC file syntax

To structure variables, for example, in the Variable Browser of ControlDesk, you can divide all model variables into hierarchical levels of subgroups. This feature is called *grouping*, see Grouping on page 9.

Refer to the following sections for information on the syntax elements of a TRC file:

- Keywords on page 14
- Variable Names on page 11
- Variable and Group Properties on page 25
- Comments on page 11

Error file

If you write your own TRC file incorrectly, an Error file is generated when you download the corresponding application: see Error File on page 12. Use this file to correct your own TRC file.

Where to go from here

Information in this section

Principles of the TRC File

Where to go from here

Information in this section

Grouping

Defining groups

For large real-time applications with numerous variables, it is useful to arrange these variables into several groups. To define a group, enclose the corresponding variables in the keywords `group` and `endgroup`. Nesting `group – endgroup` statements allows you to create multilevel tree structures. An `endgroup` statement always belongs to the most recent `group` statement. Variables that are declared between these statements belong to this group and will be listed in the Variable List of the corresponding browser node.

Naming of groups

The keyword `group` must be followed by a name enclosed in quotation marks ("…"). If quotation marks are used in the string, they must appear twice. The name must be of the same format as described in alias on page 29. If two successive slashes occur in a name (//), they are transformed into a single one.

Example

```
group   "Model"
```

```
group   "Group-Name ""A"""
```

Note

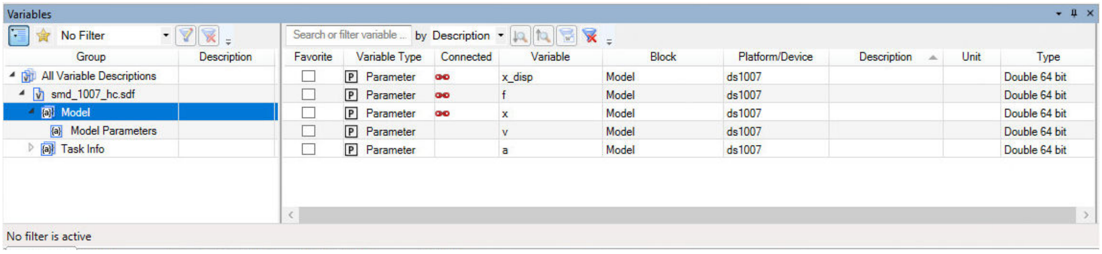
- In a TRC file, a `group` statement must always have a matching `endgroup` statement.
- Always insert an empty line between the closing brace and the `endgroup` statement.

Example

The following extract is taken from the `smd_xxxx_hc.trc` file found in the ControlDesk demo project, that is available as backup file in `<RCP_HIL_InstallationPath>\Demos\dsxxxx\GettingStarted\HandCode`.

```
group "Model"
   x_disp   flt
   f   flt
   x   flt
   v   flt
   a   flt
   group "Model Parameters"
      d   flt
      c   flt
      m   flt
   endgroup
endgroup
```

In ControlDesk's Variables controlbar, these variables will look like this:



Variables

No Filter

Group	Description
All Variable Descriptions	
smd_1007_hc.sdf	
Model	
Model Parameters	
Task Info	

Search or filter variable ... by Description

Favorite	Variable Type	Connected	Variable	Block	Platform/Device	Description	Unit	Type
□	P Parameter	🔗	x_disp	Model	ds1007			Double 64 bit
□	P Parameter	🔗	f	Model	ds1007			Double 64 bit
□	P Parameter	🔗	x	Model	ds1007			Double 64 bit
□	P Parameter		v	Model	ds1007			Double 64 bit
□	P Parameter		a	Model	ds1007			Double 64 bit

No filter is active

Note

At the end of the TRC file an empty line has to be inserted to avoid an error message caused by the TRC file parser.

Appearance of groups in ControlDesk's Variables controlbar

In ControlDesk's Variables controlbar, for example, a group will appear as a node (unless the node has the flag `HIDDEN`, see flags on page 32).

Related topics

Basics

Keywords.. 14
Variable and Group Properties... 25
Variable Names.. 11

References

Comments.. 11

Variable Names

Variable names

The name of the variable can be a scalar or an array and is limited to a maximum of 4096 characters. The name (or its alias) will appear in ControlDesk's **Variables** controlbar, for example. The name of the variable must be identical to the name of the corresponding global variable of the real-time program. Variables declared as `static` cannot be accessed, for example by ControlDesk, unless their address is explicitly given in the TRC file because such variables do not appear in the MAP file. If a variable is not defined in `<model>.c`, the line in the TRC file is accepted only if the absolute address is given.

Note

You must assign a datatype to each variable.

Example

```
X[0]
{
  type: flt
}
```

Related topics

Basics

Grouping.. 9
Keywords.. 14
Variable and Group Properties.. 25

References

alias.. 29

Comments

Syntax of a comment

TRC files may contain comments. Initial double minus characters `--` declare a line in the TRC file as a comment.

Example

```
-- this is a comment
```

Note

The length of a comment is limited to 4096 characters.

Related topics

Basics

Error File

Error messages

The experiment software parses the TRC file together with the Linker MAP file. If you write your own TRC file incorrectly, an error message will be displayed, for example, when you download the corresponding application.

Error messages are listed in the `<model>.err` or in the `<model>_user.err` file. Some of the possible error messages are:

	Error	Description
Syntax error		
	`identifier expected`	A line must begin with a name or a keyword; a `group` instruction must be followed by a name.
	`type, [or expected`	An identifier can only be followed by the given symbols.
	`type expected`	Array declarations must be followed by a type.
	`number expected`	A type can only be followed by an address or a comment.
	`illegal numeric format`	Illegal syntax used for a numeric value (decimal or hex with a leading `0x` or a trailing `h`).
	`float number expected`	A floating-point number is expected, either in absolute or exponential format.
	`extra characters`	Superfluous characters given; maybe a comment without `--.`
	`] expected`	A right bracket is expected.
	`string exceeds end of line (" expected)`	The terminating quotes of a string could not be found; multi-line strings are not allowed.
	`endgroup missing`	Each group statement requires a matching `endgroup` statement.
	`illegal endgroup`	There is no matching `group` statement for the `endgroup` statement.
	`groupname must not be empty`	The matching `group` statement must be followed by a group name in `" "`, or the description block must contain an `alias` statement.
	`filename is empty`	The keyword `_application` must be followed by a string constant that contains a file name.
	`keyword _application must not occur multiple`	The keyword `_application` may occur only once in the TRC file.
	`illegal data size`	The data size can only be `32-bit` or `64-bit` (TI floating-point data format can only be `32-bit`).
	`illegal data format`	The data format can only be `TI` or `IEEE.`
	`illegal index or array declaration`	An array must be defined in one of the following formats: 1. `[2]` 2. `[4.6]`
	`unexpected symbol`	A symbol does not fit the TRC file structure.
	`illegal use of keyword`	A keyword was not expected to be on its position.
	`string constant expected`	The keyword `_application` must be followed by a string constant.
Semantic error		
	`invalid index range`	The first index of an array declaration is higher than the last one.
	`group already defined`	A group name must not occur multiple times in the same subgroup.

Keywords

Introduction

In TRC files different keywords are used to store information on the TRC file and structure the contents.

Rules for keywords

Each keyword is optional and is followed by a string containing the corresponding value. If a keyword definition appears more than once in a TRC file, the latest definition will be applied.

> **Note**
>
> - A keyword must not be used as variable name in the real-time model.
> - All of the keywords are reserved words. You cannot use them for global variables, such as a Simulink.Parameter with *ExportedGlobal* as the storage class.
> - In a structured data type, the field names can be set to keywords, such as *value* or *default*.
> - Except for `group` and `endgroup`, all keywords are case sensitive.

Where to go from here

Information in this section

_author

Syntax

```
_author   "name"
```

Purpose

To indicate the name of the author creating the model.

Description

This keyword is used to indicate the name of the model's author. The keyword is case sensitive. The entire name must be enclosed in quotation marks ("…").

Example

```
_author      "RTI1104 7.5 (02-November-2015)"
```

_description

Syntax

```
_description   "model description"
```

Purpose

To give additional information on the model.

Description

This keyword can be used to describe the model more precisely or to add further information. The keyword is case sensitive. The entire value must be enclosed in quotation marks ("…").

Example

```
_description   "Add a clear description if possible"
```

_floating_point_type()

Syntax

```
_floating_point_type(size, format)
```

Purpose

To set a new default size for floating-point variables (flt, float).

Description

By default all types are evaluated as 32-bit variables, and floating-point values are supposed to be defined in the Texas Instruments format.

The default size of floating-point variables (`flt` or `float`, `flt*` or `float*`) can be changed using the keyword `_floating_point_type`. This keyword expects two parameters, the size (32-bit or 64-bit) and the internal format of the floating-point value, TI or IEEE.

The scope of `_floating_point_type` ranges from its current position within the TRC file until the end of file or until another `_floating_point_type` occurs.

The keyword is case sensitive.

Example

```
_floating_point_type(64, IEEE)
```

Note

The combination of 64-bit with the TI format for floating-point values is not supported and leads to an error. Variables using data types that are not allowed are removed while the MAP file is parsed.

_gendate

Syntax

```
_gendate   "date and time"
```

Purpose

To indicate the date and time when the TRC file was created.

Description

This keyword is used to indicate the date and time when the TRC file was created. The keyword is case sensitive. The entire value must be enclosed in quotation marks (“…”).

Example

```
_gendate   "05/04/2015 10:49:39"
```

_genname

Syntax

```
_genname   "name"
```

Purpose

To indicate the name of the tool generating the TRC file.

Description

This information is useful when the format of any of the blocks used in the real-time application has to be ascertained. The keyword is case sensitive. The entire value must be enclosed in quotation marks (“…”).

Example

```
_genname   "ConfigurationDesk"
```

_genversion

Syntax

```
_genversion   "number"
```

Purpose

To indicate the version of the tool generating the TRC file.

Description

If the TRC file is generated automatically, this keyword indicates the version number of the generating tool. The keyword is case sensitive. The entire value must be enclosed in quotation marks (“…”).

Example

```
_genversion   "1.2"
```

_integer_type()

Syntax

```
_integer_type(size)
```

Purpose

To set a new default size for integer variables (int) and unsigned integer variables (uint).

Description

The keyword `_integer_type` changes the default size (32-bit) of all variables defined as `int, int*` or `uint, uint*`. The size can be set to 8-bit, 16-bit, 32-bit or 64-bit. This value follows the keyword enclosed in parentheses.

The scope of `_integer_type` ranges from its current position within the TRC file until the end of file or until another `_integer_type` occurs.

The keyword is case sensitive.

Example

```
_integer_type(64)
```

_model

Syntax

```
_model   "name"
```

Purpose

To indicate the name of the model.

Description

This keyword is used to indicate the name of the model and is case sensitive. The entire value must be enclosed in quotation marks (`"…"`).

Example

```
_model   "smd_1104_sl"
```

endgroup

Syntax

```
endgroup
```

Purpose

To indicate the end of a subgroup.

Description

The keyword `endgroup` is used to close a group. Refer to Grouping on page 9.

This keyword is not case sensitive.

Example

```
...
endgroup
```

Related topics

References

group 20

endstruct

Syntax

```
endstruct
```

Purpose

To indicate the end of a structure data type.

Description

The keyword `endstruct` is used to close a structure definition in a TRC file.

This keyword is case sensitive.

Example

```
...
endstruct
```

Related topics

References

struct 22

enum

Syntax

```
enum <enumName>
{
  type: int32
  enums
    {
      <enumNumber>: <enumString>
      ...
    }
}
```

Purpose

To define enumeration values.

Description

Enums specified in MATLAB/Simulink are generated into the Variable Description File and can be used for experimentation.

Example

The definition of the enumeration values for an LED state:

```
enum LEDState
{
  type:int(32)
  enums
    {
      0:   "Off"
      1:   "Blinking"
      2:   "On"
    }
}
```

The definition of the **myLED** variable using the enum data type:

```
myLED
{
    type:  enum LEDState
    alias: "myLED"
    flags: PARAM
}
```

The definition of a type definition using an enum:

```
typedef LEDStateArray enum LEDState[2]
```

group

Syntax

```
group   "name"
```

Purpose

To define a group.

Description

The keyword `group` is used as an initialization command of a group in a TRC file. For example, in ControlDesk's Variables controlbar, this group will appear as a node. For further information please refer to Grouping on page 9.

This keyword is not case sensitive.

Example

```
group   "group_name"
{
  desc: …
```

Related topics

References

endgroup.. 18

sampling_period[daq_raster_index]

Syntax

```
sampling_period[daq_raster_index]
```

```
{
   value:
   alias:
   unit:
}
```

Purpose

To specify a DAQ raster for data capturing.

Description

To measure variables (e.g., task information variables) stored in the TRC file synchronously with a task, for example, in dSPACE's ControlDesk, the DAQ raster must be enabled for that task. The sample time of the task defines the measurement raster or DAQ raster, respectively.

A real-time application can have up to 31 tasks that have their DAQ raster enabled. A task's DAQ raster is enabled if the DAQ raster name field is not empty. For each DAQ raster one `sampling_period[daq_raster_index]` entry is located in the TRC file. The `daq_raster_index` entry represents the DAQ raster number minus 1. The index is automatically assigned during the build process.

The `alias` specifies a more intuitive name for the task. Refer to the DAQ raster name parameter of the task.

The `value/unit` pair represents the measurement raster. The `value` can be:

- The sample time of the task, if the DAQ raster is located in a periodic task.
- 0.0, if the DAQ raster is located in an asynchronous task

Note

The keyword `sampling_period[daq_raster_index]` is case sensitive.

Example

In a TRC file generated by ConfigurationDesk, the entry for the sampling period for a DAQ raster may look like this:

```
sampling_period[0]
{
   value:       0.001
   alias:       "Periodic Task 1"
   unit:        "s"
}

sampling_period[1]
{
   value:       0.01
   alias:       "Periodic Task 2"
   unit:        "s"
}
```

struct

Syntax

```
struct <structName>
```

Purpose

To define a structured data type.

Description

The keyword `struct` is used as an initialization command of a structure in a TRC file. It is followed by the type definition and must be closed with the `endstruct` keyword.

This keyword is case sensitive.

Example

```
struct MyStruct
{
  desc: …
  array-incr: -1
}
  X
  {
    type: int
    offs: -1
  }
  CustomNameForY
  {
    alias: "Y"
    type: int
    offs: -1
  }
endstruct
```

Note

If you manually create a User TRC file, you must set the values for `offs` and `array-incr` to `-1`. During the TRC file generation, these values are replaced according to the variable addresses.

Related topics

References

endstruct... 19

typedef

Syntax

```
typedef typename type[size]
```

Purpose

To define a new customized datatype.

Description

The keyword is followed by the new datatype name, the datatype being used and, enclosed in brackets, the number of elements being created. The keyword is case sensitive. The following example creates a 5 x 5 matrix.

Example

```
typedef  Seq1D  flt[5][5]
```

Note

Variables using datatypes that are not allowed are removed while the MAP file is parsed. For example, on DSP base hardware, the data types Int8 and Int16 are not supported and therefore not allowed.
Defining C code structures by means of this keyword is not possible.

Variable and Group Properties

Introduction

You can assign properties and attributes to variables or groups of variables.

Variable and group properties

For information on the naming of variables and groups, refer to Variable Names on page 11.

> **Note**
>
> - A property must not be used as variable name in the real-time model.
> - You must assign a datatype to each variable. See type (Data Type, Data Format and Type Definition) on page 38.
> - Except for the datatype, all other properties are optional.
> - Enclose the properties belonging to a variable or a group of variables in braces ({…}).

In a TRC file each variable is declared in a separate line that is followed by a block containing all properties such as the `type`, the (physical) `unit` or the `alias` of the block.

Example

```
Scalexio_PWM_PFM_In_1_MinimumFrequency
{
  type:    flt(64,IEEE)
  alias:   "Minimum frequency"
  flags:   PARAM
  range:   <0.003424; 3676470.0>
  value:   1.0
  desc:    "For further information refer to online help..."
}
```

Several property blocks

For each signal, several property blocks can be defined. Make sure that the `alias` names used for these blocks are unambiguous. Defining several property blocks is useful whenever a signal should be observed with different data types.

Example The following example shows how the signal `myUnion` can be made accessible both as an integer value and as a float value for experiment software.

```
myUnion
{
  alias:  "Output as int"
  type:   int
    …
}
```

```
myUnion
{
  alias:  "Output as float"
  type:   float
    …
}
```

Where to go from here

Information in this section

addr

Syntax

```
addr:   address
```

Purpose

To specify the memory address of a variable that is not accessible via the MAP file.

Description

If a variable is allocated to an absolute address outside the scope of the linker (for example, in dual-port memory) this variable does not appear in the MAP file. However, it can be made accessible, for example, for ControlDesk, if the base address of this variable is known. Therefore, this address in the real-time processor's memory has to be entered.

> **Note**
>
> The `addr` property is not available for references.

Addresses may be declared as:

- Absolute addresses in hexadecimal notation starting with `0x`
- Absolute addresses in hexadecimal notation with at least one leading digit and a trailing `h` character
- Absolute addresses in decimal notation

For example, the address can be written as follows:

Variable Name	Data Type	Address Notation
`X[0]`	`flt`	`0x805000`
`X[0]`	`flt`	`805000h`
`X[0]`	`flt`	`8409088`

Note

Although the given variable is declared with an address it is not a pointer variable. Its type remains `float`. This is in contrast to the `float *` type, which means that a pointer to a float is located at the address.

Arbitrary array subranges can be referenced in TRC files as shown below. Each array element is treated, for example, by ControlDesk, as a separate variable. Array indices in TRC files as well as in C programs always start at zero. The variable `rtB[3]` is equal to the fourth element of that array.

Although only the base address has to be given, the offset of the variable's address will be calculated automatically. For example:

```
rtB[3]
{
  type:  flt
  alias: "Element with index 3"
  addr:  0x00000010
}
```

The following type of declaration must be used to access arrays that were allocated during run time by function calls to `malloc()` or `calloc()`. For example:

```
x_dot[0..3]
{
  type:  float *
  alias: "Array access"
}
```

Note

It is not possible to declare an array of pointers in the TRC file. An array with a pointer datatype (`float *`, `int *` or `uint *`) means that there is a pointer variable in the C program pointing to an array of `float`, `int or uint` values. The subsequent array indices in the TRC file are used to access the respective array elements.

alias

Syntax

```
alias:   "customized variable name"
```

Purpose

To define a more intuitive name for a variable.

Description

This property can be used to set the **`alias`** name of a variable (array element or scalar variable) that has already been defined in order to provide the variable with a more intuitive name. Alias names can have two formats: either a standard C variable name or a string. The name of a variable is formed by an underline or a letter ('_', 'a-z' or 'A-Z') as the first character, followed by a sequence of alphanumeric characters ('_', 'a-z', 'A-Z', or '0-9'). A string begins and ends with quotation marks (").

Example

```
X[0]
{
  alias:  "rpm"
  addr:   0x805000
  …
}
```

If two successive slashes occur in a name (//), they are transformed into a single one. If quotation marks are used in the string, they must appear twice.

Example

```
"This is block ""a""".
```

Note

- The **`alias`** name must defined within the braces of the corresponding variable. The variable cannot be renamed in another property block.
- Alias names are limited to a maximum of 128 characters.

array-incr

Syntax

```
array-incr:   number
```

Purpose

To specify the memory size in bytes of the related structure.

Description

In a user TRC file, the array increment is to be specified by -1. During the build, the value is replaced according to the platform-specific variable addresses. This property is mandatory and valid only for structure definitions.

Example

```
struct MyStruct
{
  desc: …
  array-incr: -1
}
  X
  {
    type: int
    offs: -1
  }
  CustomNameForY
  {
    alias: "Y"
    type: int
    offs: -1
  }
endstruct
```

bitmask

Syntax

- `bitmask:   hexnumber`
- `bitmask:   startbit : endbit`

Purpose

To mask bits of the signal value.

Description

This property provides bit access to the signal value. The least significant bit is defined as bit number 0.

Example

- `bitmask:  0xF012`
- `bitmask:  8:11`

block

Syntax

```
block:   "blocktype"
```

Purpose

To describe the blocktype of a Simulink block.

Description

This property can only be assigned to nodes. For example, it stores the type of a Simulink block that is represented by this node.

Example

```
block:   "Gain"
```

default

Syntax

```
default:   value
```

Purpose

To specify the default value for a signal.

Description

This property specifies the default value for a signal, which can automatically be displayed, for example, in a ControlDesk instrument. The permissible values depend on the type of the signal. String values must be enclosed in quotation marks (").

Example

```
default:   75.2
```

desc

Syntax

```
desc:   "text"
```

Purpose

To describe a signal or a group.

Description

This field contains text describing a signal or a group.

Example

```
desc:   "Current_Speed"
```

flags

Syntax

```
flags:   flag  [ | flag ]
```

Purpose

To describe special properties of a signal.

Description

This field contains flags describing special properties of the signal. Flags can be combined and also be set to variables or blocks.

Example

```
flags:   HIDDEN  |  PARAM
```

The following table lists all available flags alphabetically:

Flag	Purpose
HIDDEN	To hide a node in ControlDesk's Variables controlbar.
OUTPUT	To mark block outputs.
PARAM	To mark a variable as a parameter.
READONLY	To make a variable read-only. The variable cannot be written.
DEPRECATED	To mark an item as deprecated.

increment

Syntax

```
increment:  time_in_seconds
```

Purpose

To specify the unit increment for a task.

Description

The increment value corresponds to the sampling period of the model.

Example

```
increment:  0.01
```

offs

Syntax

```
offs:   no_of_bytes
```

Purpose

To specify the offset of a field definition in a structure in bytes.

Description

In a user TRC file, the offset is to be specified by -1. During the build, the value is replaced according to the platform-specific variable addresses. This property is mandatory and valid only for struct elements.

Example

```
struct MyStruct
{
  desc: …
  array-incr: -1
}
  X
  {
    type: int
    offs: -1
  }
  CustomNameForY
  {
    alias: "Y"
    type: int
    offs: -1
  }
endstruct
```

origin

Syntax

```
origin:   "model/subsystem/.../block/signal"
```

Purpose

To specify the entire path of signals, parameters and blocks.

Description

In TRC files generated by ConfigurationDesk, this property is used for signal labels to indicate the path of the corresponding signal in the Simulink model.

Example

```
origin:  "smd_1007_sl/Integrator 1/Out1"
flags:   LABEL|READONLY
```

range

Syntax

```
range:   <min; max>
```

Purpose

To define the valid range for the signal value variation.

Description

Integer, floating point and exponential numbers are possible for min and max. Use the keyword `inf` to define an infinite limiting value.

Example

- `range:  <-5; 5>`
- `range:  <-5; inf>`

refelem

Syntax

```
refelem:   "elementname"
```

Purpose

To specify a field in a structure or an element in an array that is used as reference.

Description

If the `refvar` property references a structure or an array, the `refelem` property specifies the concrete element to be used as a reference.

The element name is specified as a string.

A nested structure is described with dots as path delimiters. The path must then also start with a dot.

For an array element, only the index of the element is given in square brackets. The name of the array is specified in the related `refvar` property.

When using arrays in structures or arrays of structures, the notations for referencing a structure field and for referencing an array element can be combined, e.g., `refelem: ".myStructField[2].myArray[5]"`.

Example

Example of a structure element

```
struct MyStruct
{
  desc: …
  array-incr: -1
}
  X
  {
    type: int
    offs: -1
  }
  CustomNameForY
  {
    alias: "Y"
    type: int
    offs: -1
  }
endstruct
```

```
pointStructVar
{
  type:  struct pointStruct
  alias: "MyPointStructVar"
  flags: PARAM
}

ref2FieldVar
{
  alias:    "MyFieldVar"
  refgroup: "."
  refvar:   "MyPointStructVar"
  refelem:  ".X"
}
```

Example of an array

```
typedef IntArray int[5]

intArrayVar
{
  type: IntArray
  alias: "MyIntArray"
  flags: PARAM
}

ref2IntArrayElem
{
  alias:   "MyArrayElem"
  refgroup: "."
  refvar:   "MyIntArray"
  refelem:  "[2]"
}
```

refgroup

Syntax

```
refgroup:   "groupname"
```

Purpose

To specify the group name of a variable reference.

Description

The refgroup property specifies in which group the referenced variable is declared. The group name is specified as a string and contains either an absolute or a relative path to the group.

The following elements are supported in the path definition:

- `"/"` as the path delimiter
- `".."` to specify the parent element
- `"."` to specify the current element

Absolute paths have to begin with a slash (/), relative paths can begin with the name of a subgroup, with a single dot (.), or with two dots (..).

The `refgroup` property is mandatory for a reference element. The definition of the referenced group can be placed before or after the definition of the referenced variable in the variable description file.

Example

Example for an absolute path:

```
refgroup: "/Tunable Parameters"
```

Example for a relative path:

```
refgroup: "../MySubGroup/MyNestedSubGroup"
```

refvar

Syntax

```
refvar:   "variablename"
```

Purpose

To specify the variable name of a reference.

Description

The `refvar` property is used within a `reference` element and requires at least a related `refgroup` property. The variable name is specified as a string. If an `alias` is specified, the alias name is used, otherwise the name of the referenced variable is used.

Example

Example of specifying the `refvar` property with an alias defined for the variable.

```
typedef IntArray int[5]

intArrayVar
{
  type: IntArray
  alias: "MyIntArray"
  flags: PARAM
}

ref2IntArrayElem
{
  refgroup: "."
  refvar:   "MyIntArray"
  refelem:  "[2]"
}
```

scale

Syntax

```
scale:   [numerator polynomial] / [denominator polynomial]
```

Purpose

To convert the signal value.

Description

When you read the signal from a data source, the signal value is converted by using the scale function. The value conversion is an option and not performed automatically.

The denominator polynomial is optional. Possible coefficients are integer, floating point and exponential numbers.

Example

- `scale:  [2 0 3] / [2 4]`
- `scale:  [2, 0, 3] / [2, 4]`
- `scale:  [2, 1.345, 2^-11]`

scaleback

Syntax

```
scaleback:   [numerator polynomial] / [denominator polynomial]
```

Purpose

To reverse the scale function.

Description

When you write the value to a data source, the value is converted to the signal value by using the scaleback function. The value conversion is an option and not performed automatically.

The denominator polynomial is optional. Possible coefficients are integer, floating point and exponential numbers.

Example

- `scaleback:  [2 4] / [2 0 3]`
- `scaleback:  [2, 4] / [2, 0, 3]`
- `scaleback:  [2, 1.345, 2^-11]`

type (Data Type, Data Format and Type Definition)

Syntax

```
type:   type (size,format)
```

Purpose

To specify the type, format and size of a variable and to define look-up tables.

Description

- The size has to be set according to the real-time hardware. The following table displays the permissible data types and sizes:

Data Type	Description
`int (8)`	8-bit integer value
`int (8) *`	pointer to an 8-bit integer value
`int (16)`	16-bit integer value
`int (16) *`	pointer to a 16-bit integer value
`int (32)`	32-bit integer value
`int (32) *`	pointer to a 32-bit integer value
`int (64)`	64-bit integer value
`int (64) *`	pointer to a 64-bit integer value
`uint (8)`	8-bit unsigned integer value
`uint (8) *`	pointer to an 8-bit unsigned integer value
`uint (16)`	16-bit unsigned integer value
`uint (16) *`	pointer to a 16-bit unsigned integer value
`uint (32)`	32-bit unsigned integer value

Data Type	Description
`uint (32) *`	pointer to a 32-bit unsigned integer value
`uint (64)`	64-bit unsigned integer value
`uint (64) *`	pointer to a 64-bit unsigned integer value
`flt (32, IEEE)`	32-bit floating-point value
`flt (32, IEEE) *`	pointer to a 32-bit floating-point value
`flt (64, IEEE)`	64-bit floating-point value
`flt (64, IEEE) *`	pointer to a 64-bit floating-point value

- The format for floating-point values can only be IEEE standard. If you specify a variable of integer type, you do not need to define the format.
- You can additionally specify a variable of array, enumeration or struct type:
  - Arrays on page 39
  - Enumerations on page 40
  - Structs on page 40

Note

The `type` property used in structure elements does not support pointer types.

Arrays

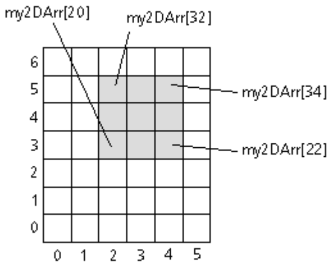
Description

- If you defined a 6x7 matrix, you can refer to specific elements of this two-dimensional array, for example:

  `my2DArr[2..4][3..5] float (32,IEEE)`

  This line refers to the following nine elements of `my2DArr`: `my2DArr[20]` … `my2DArr[22]`, `my2DArr[26]` … `my2DArr[28]` and `my2DArr[32]` … `my2DArr[34]`

  The following illustration shows the 6x7 matrix `my2DArr`. The selected matrix elements `my2DArr[2..4][3..5]` are highlighted.



Note

The `alias` name(s) are also added with index information in order to be unambiguous. Always start with zero when you count the elements of an array.

- You can also create an n-dimensional look-up table. Insert `lookup` between the `typename` and the `type` within a `typedef` statement.

Syntax `typedef typename lookup type`

Example `typedef Lookup2D lookup flt[6][4]`
`MyLookupTable Lookup2D`

For information on how to define new datatypes, refer to typedef on page 23.

Enumerations

Description Enums specified in MATLAB/Simulink are generated into the Variable Description File and can be used for experimentation.

Syntax

```
enum <enumName>
{
  type: <Integer-DataType>
  enums
    {
      <enumNumber>: <enumString>
      ...
    }
}
```

Example The definition of the enumeration values for an LED state:

```
enum LEDState
{
  type:int(32)
  enums
    {
      0:   "Off"
      1:   "Blinking"
      2:   "On"
    }
}
```

The definition of the `LEDState` variable using the enum data type:

```
LEDState
{
    type:  enum LEDState
    alias: "LEDState"
    value: 2
    unit:  "-"
}
```

The definition of a type definition using an enum:

```
typedef LEDStateArray enum LEDState[2]
```

Structs

Description Structs specified in MATLAB/Simulink are generated into the Variable Description File and can be used for experimentation.

Syntax

```
struct <structname>

{
  desc: <String>
  array-incr: <Integer-DataType>
}
      <StructElement>
      {
         type: <String-DataType>
         offs: <Integer-DataType>
      }
      ...
endstruct
```

Example The definition of the struct elements:

```
struct MyStruct
{
  array-incr: -1
}
  structField0
  {
    alias: "element1"
    type: flt(64,IEEE)
    offs: -1
    unit: "mph"
    range; < 0.0 ; 225.0 >
    desc: "SPeed of the vehicle."
  }
  structField1
  {
    alias: "element2"
    type: int(8)
    offs: -1
  }
endstruct
```

Example of a type definition using a struct:

```
typedef PointStructArrayType struct MyStruct[4]
```

unit

Syntax

```
unit:   "physical_unit"
```

Purpose

To set the physical unit for a signal value.

Description

This property gives information about the physical unit of the signal value. This text can automatically be displayed, for example, in the caption of an instrument.

Example

```
unit:   "mph"
```

value

Syntax

```
value:   value
```

Purpose

To specify the initial value of a parameter.

Description

In ControlDesk, this property is mainly used by the data set management. The specified value is used when initial data sets are generated. For details on data sets in ControlDesk, refer to Data Sets and their Relation to Memory Pages (ControlDesk Calibration and Data Set Management 📖).

Example

```
value:   75.2
```