Data Access Run-Time
Library Reference Manual

Order Number: MDSplus—DARTL001

February 2, 1993

This manual describes the routines used to access MDSplus model and pulse files. These Data Access Run-Time Library routines are found in the SYS$SHARE:MDSLIB.EXE shared image.

Revision/Update Information: This is a new manual.
Operating System and Version: VMS Version 5.2 or higher
Software Version: MDSplus Version 1.0 Field Test
### TABLES

<table>
<thead>
<tr>
<th>1</th>
<th>MDS$ Routines</th>
<th>9</th>
</tr>
</thead>
</table>

Preface

Intended Audience

This Manual is intended for all users of the MDSplus system.

Document Structure

This document contains descriptions of all the MDS$ routines used for accessing the Model and Pulse files within user written data analysis programs. Most arguments to these routines are passed by descriptor enabling the user to specify the data types and size desired in the transfer of information to and from the MDSplus data files. The passed by descriptor mechanism is well supported by the VAX FORTRAN compiler with syntax provided to automatically build VMS descriptors. The following is a description of the topics covered in this manual.

• Introduction—Provides a general overview of the MDS$ routines including examples of there use.
• Routine Summary—Describes the various categories of routines available.
• Routine Descriptions—Provides detailed descriptions of the routines including argument descriptions and argument passing mechanisms.
Data Access RTL Introduction

There are several routines contained in the MDSLIB shared image that access the Model and Pulse files. Using these routines, a MDSplus user can read and write data into the experiment model and pulse files. The user only needs to learn the calling format of a few routines to utilize the full functionality of the interface to MDSplus data, including the built-in expression handling capabilities of MDSplus.

The calling formats of the routines are described in detail in Part II of this document. The documentation format resembles the VMS routine documentation. For a description of the various sections of the routine documentation see the VMS Manual entitled "Introduction to System Routines" manual in Volume 3 of the Programming documentation set.
Only a few subroutine calls are required to access data stored or to store more data in an MDS database file. To read in data, the user only needs to open the database file using the MDS$OPEN routine, read the desired data into a program variable or array using the MDS$VALUE routine, and then close the file using the MDS$CLOSE routine. Likewise, to write data to the database file the user calls MDS$OPEN to open the file, MDS$PUT to write the data, and MDS$CLOSE to close the file.

MDS can handle many different data types and various shapes of data from simple scalar values to multi-dimensional arrays to special MDS structures such as signals, which contain both array values and independent axis information. When reading data from MDS into your local program variable, MDS automatically converts the data to match the type of your local variable. When calling an MDS routine that moves data to or from a local variable, you must use a special argument passing mechanism called "pass by descriptor". When passing an argument by descriptor, instead of passing the data itself to the subroutine, a small block of information is passed, which tells the subroutine the kind of data being passed, the size of the data, its dimensions, and the address of the actual data. If you are using the VAX/VMS FORTRAN language, it is very easy to pass variables by descriptor. FORTRAN provides a built-in function, %DESCR(variable), which constructs this block of information for you. If you are using a language other than FORTRAN, you should read about argument descriptors in the "Introduction to VMS System Routines" manual of the VMS programming documentation set.

There are only a few other routines we haven't mentioned yet that may be useful when accessing MDS databases. One of these is the MDS$SET_DEFAULT routine. MDS databases are structured hierarchically. This structure, similar to directory and subdirectory structures of a file system, makes it easy to organize the data in the database. However, like directories in a file system, you must completely specify the path for locating the data you are interested in. Often you are interested in several pieces of data, which are located nearby in the database structure and it would be cumbersome to specify the full path name of each piece of data each time you access it. The MDS$SET_DEFAULT routine lets you specify a location of the database to use as a starting point when accessing data. After calling MDS$SET_DEFAULT, you only need to specify paths relative to that location when accessing data.

Two other routines, MDS$GET and MDS$FREE, are more advanced and will be discussed later. Their primary use is to allow more efficient access to data when more than one piece of information is to be extracted from a single item in the database.
Reading data

The easiest way to describe how to access data in an MDS database is to give an example. The following example simply opens an MDS database file, reads in an array of data, and closes the file. In the example the database name is called "CMOD", the data is for shot 42, and the database item or node containing the data is called "\ TOP.XRAY:CHANNEL_1". You would normally replace, "CMOD", 42, and "\ TOP.XRAY:CHANNEL_1" with the database, shot, and node that match your experiment.

Example 1  Reading data - simple FORTRAN example

```fortran
Real VOLTS(8192)
Integer SAMPLES
CALL MDS$OPEN('CMOD',42)
CALL MDS$VALUE('\TOP.XRAY:CHANNEL_1',%DESCR(VOLTS),SAMPLES)
CALL MDS$CLOSE
END
```

In the above example, the data in the node "\TOP.XRAY:CHANNEL_1" is converted to floating point and loaded into the user's variable "VOLTS". The number of array elements in "\TOP.XRAY:CHANNEL_1" is returned in the user's variable "SAMPLES". If this number was less than 8192, the size of the "VOLTS" variable, the remaining elements of the "VOLTS" array will contain zero's.

The previous example returned the data from a single node in the tree. You can also specify an expression where the node name was specified. The next example shows the how an expression might be used:

Example 2  Reading data - expression

```fortran
Real AMPS(8192)
Integer SAMPLES
CALL MDS$OPEN('CMOD',42)
CALL MDS$VALUE('\TOP.XRAY:CHANNEL_1 * 12.0356 + 5.9876',%DESCR(AMPS),SAMPLES)
CALL MDS$CLOSE
END
```

The same three subroutine calls are used in the above example as in Example 1 but in this case the data is multiplied by a constant and an offset is added to the result before returning the answer into the user's variable. Instead of using the constants in the expression string, the user could have referenced other nodes in the database or other variables in the program. A special syntax is provided in MDS expressions for substituting user variables as arguments in expressions. This is done by specifying a "$" followed by an optional argument index ($,$1,$2 etc.). The actual user variables must immediately follow the expression string in the subroutine call and they must be passed by descriptor. Example 3 demonstrates the use of expression arguments.
Example 3  Reading data - expression with arguments

```fortran
Real AMPS(8192)
Integer SAMPLES
Real COEF /12.0356/
Real OFFSET /5.9876/
CALL MDS$OPEN('CMOD',42)
CALL MDS$VALUE('TOP.XRAY:CHANNEL_1 * $1 + $2,%DESCR(COEF),%DESCR(OFFSET), + %DESCR(AMPS),SAMPLES)
CALL MDS$CLOSE
END
```

The above example should return exactly the same answer as in Example 2 but the coefficient and offset are now contained in user variables.

Some records stored in an MDS database are more complex than simple scalar values or arrays of values. Signals stored by digitizers, for example, may contain much more information than the count values read in from the digitizer channel. Often, units, engineering conversion, and independent axis information is contained in the signal in addition to the raw data count values. Assuming "TOP.XRAY:CHANNEL_1" is a signal stored by a digitizer we can illustrate how one might get more than one piece of information from this node in the database file:

Example 4  Reading data - signals

```fortran
Real VOLTS(8192)
Real TIME(8192)
Character*10 VOLT_UNITS
Integer VU_LENGTH
Character*10 TIME_UNITS
Integer TU_LENGTH
Integer*2 COUNTS(8192)
Integer SAMPLES
CALL MDS$OPEN('CMOD',42)
CALL MDS$VALUE('TOP.XRAY:CHANNEL_1',%DESCR(VOLTS),SAMPLES)
CALL MDS$VALUE('UNITS(TOP.XRAY:CHANNEL_1)',%DESCR(VOLT_UNITS),VU_LENGTH)
CALL MDS$VALUE('RAW(TOP.XRAY:CHANNEL_1)',%DESCR(COUNTS),SAMPLES)
CALL MDS$VALUE('DIM_OF(TOP.XRAY:CHANNEL_1,0)',%DESCR(TIME),SAMPLES)
CALL MDS$VALUE('UNITS(DIM_OF(TOP.XRAY:CHANNEL_1,0))',%DESCR(TIME_UNITS),TU_LENGTH)
cALL MDS$CLOSE
END
```

In Example 4 the digitizer channel data converted to volts is returned in variable "VOLTS", the units are returned in variable "VOLT_UNITS", the raw data is returned in variable "COUNTS", the time stamps for the data is returned in variable "TIME", and the units of time are returned in variable "TIME_UNITS". Note that in the calls to MDS$VALUE containing the "VOLT_UNITS" and "TIME_UNITS" variables, the %DESCR() is not required since strings are passed by descriptor by default for VAX FORTRAN.

There is a much more efficient way of obtaining the above information however. In the above example, the same signal is accessed five times. Each access might represent multiple system I/O's. Some of these I/O's might be buffered, that is, they might just be accessing physical RAM
memory or more likely they might end up doing actual disk I/O's. To avoid, fetching the same data over and over again, MDS provides a mechanism for fetching the data into your program's local memory and then extracting the various pieces of information from the local copy instead of reading it again from the database. To do this you must use the MDS$GET routine. This routine returns a special value called a "data-id", which identifies a block of memory containing the actual data. You then use this "data-id" in calls to MDS$VALUE in place of user variables. The "data-id" must be passed "by value" instead of "by descriptor". In FORTRAN you use the %VAL() built-in instead of %DESCR(). The "data-id" must be zero before calling MDS$GET. If it is non-zero, MDS will think that the data-id was returned from a previous call to MDS$GET and it will attempt to free the memory of this data-id before allocating more for this call. Normally you should not change the value contained in a "data-id" variable except setting its initial value to zero at the beginning of a program. You can re-use a "data-id" in subsequent calls to MDS$GET and the virtual memory handling will be performed automatically for you. The following example performs the same function as Example 4 but much more efficiently.

**Example 5  Reading data - using data-id's**

```fortran
Real VOLTS(8192)
Real TIME(8192)
Character*10 VOLT_UNITS
Integer VU_LENGTH
Character*10 TIME_UNITS
Integer TU_LENGTH
Integer*2 COUNTS(8192)
Integer SAMPLES
Integer DATA_ID
Integer T_DATA_ID
CALL MDS$OPEN('CMOD',42)
DATA_ID = 0
CALL MDS$GET('\TOP\XRAY:CHANNEL_1',DATA_ID)
CALL MDS$VALUE('$',%VAL(DATA_ID),%DESCR(VOLTS),SAMPLES)
CALL MDS$VALUE('UNITS($)',%VAL(DATA_ID),VOLT_UNITS,VU_LENGTH)
CALL MDS$VALUE('RAW($)',%VAL(DATA_ID),%DESCR(COUNTS),SAMPLES)
T_DATA_ID = 0
CALL MDS$GET('DIM_OF($,0)',%VAL(DATA_ID),T_DATA_ID)
CALL MDS$FREE(DATA_ID)
CALL MDS$VALUE('$',%VAL(T_DATA_ID),%DESCR(TIME),SAMPLES)
CALL MDS$VALUE('UNITS($)',%VAL(T_DATA_ID),TIME_UNITS,TU_LENGTH)
CALL MDS$FREE(T_DATA_ID)
CALL MDS$CLOSE
END
```

Note that data-id's can be used as replacement arguments in expressions much the same way as user variables with the only difference being the way they are passed to the MDS$VALUE routine (%VAL() instead of %DESCR()). This example also introduces the use the MDS$FREE routine not seen before. The MDS$FREE routine will free the memory associated with a data-id and set the data-id variable to zero. You should always free data-id's when you are done using them.
The above examples demonstrate just about all there is to know about accessing data in MDS databases. For more information on the syntax of MDS expressions and a complete list of operators and functions see the "MDS Expression Evaluator" manual.

3 Writing data

Writing data into MDS databases is a simple as reading data. The only additional routine you need to know is the MDS$PUT routine. The put routine takes the name of the node where you want to write data and an expression with optional replacement arguments describing what to write into the node. The following is an example of the simplest case where the contents of a FORTRAN variable is written to a node in the database:

Example 6 Writing data - FORTRAN variable

```fortran
SUBROUTINE PUTANS(ANSWER)
  Real ANSWER
  CALL MDS$OPEN('CMOD',42)
  CALL MDS$PUT('TOP.XRAY:FACTOR','$',%DESCR(ANSWER))
  CALL MDS$CLOSE
  RETURN
END
```

This example simply puts a floating point number into the node "TOP.XRAY:FACTOR". You might want to put something more complex into the node than a number or array. There are many "BUILD" built-in functions that the expression evaluator knows about for building almost all kinds of higher level data structures supported by MDS. One type of complex data you can write into MDS is an expression. The following example would load an expression in the database:

Example 7 Writing data - expression

```fortran
SUBROUTINE PUTANS(FACTOR)
  Real*8 FACTOR
  CALL MDS$OPEN('CMOD',42)
  CALL MDS$PUT('TOP.XRAY:CHAN_1_FCT','TOP.XRAY:CHANNEL_1 * $1',%DESCR(FACTOR))
  CALL MDS$CLOSE
  RETURN
END
```

The above example would store an expression in the node called "TOP.XRAY:CHAN_1_FCT". The MDS$PUT routine does not evaluate the expression it only compiles the expression into an internal representation and stores this representation into the database file. If you wanted the expression evaluated first you can use the MDS$GET routine and then write the answer using the data-id.
Example 8  Writing data - using data-id

```fortran
SUBROUTINE PUTANS(FACTOR)
  Real*8    FACTOR
  Integer   DATA_ID
  CALL MDS$OPEN('CMOD',42)
  DATA_ID = 0
  CALL MDS$GET('TOP.XRAY:CHANNEL_1 * $1',%DESCR(FACTOR),DATA_ID)
  CALL MDS$PUT('TOP.XRAY:CHAN_1_FCT','$',%VAL(DATA_ID))
  CALL MDS$FREE(DATA_ID)
  CALL MDS$CLOSE
RETURN
END
```

In this example the expression in the MDS$GET call would be evaluated and returned as a data-id in the "DATA_ID" variable. The MDS$PUT routine then would store the result of this expression. Note that we purposely did not use MDS$VALUE to evaluate the expression. If we had used MDS$VALUE to compute an answer into a local FORTRAN array before writing, we might have altered the data type and/or shape of the result of the expression. If "TOP.XRAY:CHANNEL_1" were a signal, for example, the units and axis information would have been stripped off.

4  Routine Summary

The interface to the MDS database files is quite simple while providing the full capabilities of powerful MDS Expression Evaluator. You need to learn only a few routines to access data in any MDS database file from within your programs. The following table list these routines:
Table 1  MDSS Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS$CLOSE</td>
<td>Close a model or pulse file</td>
</tr>
<tr>
<td>MDS$FREE</td>
<td>Free the memory associated with a data-id</td>
</tr>
<tr>
<td>MDS$GET</td>
<td>Get data (into data-id)</td>
</tr>
<tr>
<td>MDS$OPEN</td>
<td>Open the experiment model or pulse file</td>
</tr>
<tr>
<td>MDS$PUT</td>
<td>Put data in model or pulse file</td>
</tr>
<tr>
<td>MDS$SET_DEFAULT</td>
<td>Change current default node</td>
</tr>
<tr>
<td>MDS$VALUE</td>
<td>Get data (into program variable)</td>
</tr>
</tbody>
</table>

These routines are documented in detail in the reference section of this manual.
Data Access Routine Descriptions

This part contains complete reference descriptions of the routines used for accessing MDSplus Model and Pulse files.
MDS$CLOSE—Close an pulse file

The MDS$CLOSE routine will close one or all opened experiment model or pulse files.

**FORMAT**

\[
\text{status} = \text{MDS$CLOSE}([\text{experiment,shot-number}])
\]

**RETURNS**

<table>
<thead>
<tr>
<th>VMS Usage</th>
<th>cond_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>longword (unsigned)</td>
</tr>
<tr>
<td>access</td>
<td>write only</td>
</tr>
<tr>
<td>mechanism</td>
<td>by value</td>
</tr>
</tbody>
</table>

**ARGUMENTS**

- **experiment**
  - VMS Usage: char_string
  - type: character string
  - access: read only
  - mechanism: by descriptor
  - Argument specifying the experiment and optional sub-tree name. The experiment model and pulse files are located using the experiment name and shot number. When referencing a sub-tree, append the sub-tree name to the experiment name separating the experiment and sub-tree with a dollar character (i.e., experiment$sub-tree).

- **shot-number**
  - VMS Usage: longword
  - type: longword
  - access: read only
  - mechanism: by reference
  - Argument specifying the shot number. Two shot numbers are reserved for referencing the current shot number and the experiment model.
    - Shot 0 — current shot for experiment
    - Shot -1 — experiment model

**DESCRIPTION**

MDS$CLOSE closes an opened tree. If both the experiment and shot-number argument are omitted, all open MDSplus database files will be closed. You must specify both the experiment and shot-number or use no arguments at all.

**RETURN VALUES**

- TREE$_NORMAL Operation successfully completed
- TREE$_NOT_OPEN No trees currently open
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE$_INVTREE</td>
<td>Internal memory pointers have been corrupted in your program.</td>
</tr>
<tr>
<td>RMS$_xxxxxx</td>
<td>Any error returned from the VMS record management system.</td>
</tr>
</tbody>
</table>
MDS$FREE—Free a Data-id

The MDS$FREE routine frees the memory associated with a data-id.

**FORMAT**

\[ \text{status} = \text{MDS$FREE}(\text{data-id}) \]

**RETURNS**

VMS Usage: cond_value
- type: longword (unsigned)
- access: write only
- mechanism: by value

**ARGUMENTS**

\( \text{data-id} \)
- VMS Usage: context
- type: longword
- access: modify
- mechanism: by reference

Argument containing a data-id that was returned by a call to MDS$GET. The data-id argument is the address of an unsigned longword integer containing that value.

**DESCRIPTION**

The MDS$FREE routine frees the memory associated with a data-id. A data-id is an identifier returned by the MDS$GET routine that can be used in subsequent MDS$GET, MDS$PUT, or MDS$VALUE routines. See the description of the MDS$GET routine for more information on the use of data-id's.

**RETURN VALUES**

- SS$_NORMAAL: Routine successfully completed.
- LIB$_INVSTRDES: A descriptor passed to a general library procedure did not contain a valid DSC$B_CLASS field.
- LIB$_BADBLOADR: argument contained a bad block address. An address was outside the area allocated or part of the space being deallocated was previously deallocated.
Data Access Routines
MD$GET

MD$GET—Get data

The MD$GET routine returns a data identifier that contains the internal representation of the data requested.

**FORMAT**

\[
\text{status} = \text{MD$GET}(\text{expression [, arg1, ..., argn] , data-id})
\]

**RETURNS**

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

**ARGUMENTS**

**expression**

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor
Argument specifying an MDSplus expression. The expression can be as simple as a node name or as complex as an arithmetic expression containing any number of node names, constants or argument placeholders. See the MDSplus Expression Evaluator manual for detailed syntax and functionality of MDSplus expressions.

**argn**

VMS Usage: value
type: any
access: read only
mechanism: by descriptor
Argument specifying a replacement argument to be inserted in the expression in place of argument placeholders (i.e., $,$1,$2,$n). These optional arguments permit the user to insert the value of local variables in the evaluation of MDSplus expressions. Pass local variables to this routine by descriptor. There is one special exception to this passing mechanism which is in the use of data-id's returned by the MD$GET routine. When using a data-id, you must pass the data-id by value.

**data-id**

VMS Usage: context
type: longword
access: modify
mechanism: by reference
The data-id argument is the address of an unsigned longword integer to receive the identifier. The data-id contents should be zero when first calling MD$GET. If there is a non-zero value, MD$GET will assume that the value is an identifier returned from a previous call to MD$GET and the memory associated with the identifier will be freed before a new identifier is returned.
**DESCRIPTION**

The MDS$GET routine will return a data identifier that contains an internal representation of the evaluated expression provided. Since MDSplus data items may contain several related pieces of information, (i.e., value, units, independent axes information), it is much more efficient to use MDS$GET to retrieve all the information into a data-id and then use MDS$VALUE to retrieve the individual parts than to use MDS$VALUE alone. See the following example.

**RETURN VALUES**

- **TREE$_NORMAL** - Operation successfully completed
- **TREE$_NOT_OPEN** - No trees currently open
- **TREE$_INVTREE** - Internal memory pointers have been corrupted in your program.
- **RMS$_xxxxxx** - Any error returned from the VMS record management system.

**EXAMPLES**

The following examples show two ways to access the same data. The second example is far more efficient than the first example because it requires much less I/O and computation.

```
1
Integer*4 ID
Integer*2 IDATA(8192)
Real*4 TIME(8192)
Integer*4 DLENGTH
Character*20 UNITS
Integer*4 ULENGTH
CALL MDS$OPEN('CMOD',0)
CALL MDS$SET_DEFAULT('.POWER.MAGNETS')
CALL MDS$VALUE('TOROIDAL',%DESCR(IDATA),DLENGTH)
CALL MDS$VALUE('UNITS(TOROIDAL)',UNITS,ULENGTH)
CALL MDS$VALUE('DIM_OF(TOROIDAL,0)',%DESCR(TIME),TLENGTH)
CALL MDS$CLOSE
```

In the above example, the value, units, and time-stamps are returned for the signal called "TOROIDAL". However, in obtaining this information the signal was retrieved from the database three times. This could result in many physical I/O's, depending on the complexity of the signal and how many other nodes are referenced within the signal. A much more efficient method of obtaining the same information follows:

```
2
Integer*4 ID
Integer*2 IDATA(8192)
Real*4 TIME(8192)
Integer*4 DLENGTH
Character*20 UNITS
Integer*4 ULENGTH
CALL MDS$OPEN('CMOD',0)
CALL MDS$SET_DEFAULT('.POWER.MAGNETS')
CALL MDS$GET('TOROIDAL',ID)
CALL MDS$VALUE('TOROIDAL',%VAL(ID),%DESCR(IDATA),DLENGTH)
CALL MDS$VALUE('UNITS(TOROIDAL)',%VAL(ID),UNITS,ULENGTH)
CALL MDS$VALUE('DIM_OF(TOROIDAL,0)',%VAL(ID),%DESCR(TIME),TLENGTH)
CALL MDS$CLOSE
```

Note that in this example the signal is obtained from the database only once in the call to MDS$GET. Subsequent calls to MDS$VALUE only
Data Access Routines

MDS$GET

extract the pieces of information about the signal from the copy residing in local memory referenced by the data-id.
MDS$OPEN—Open File

The MDS$OPEN routine will open an experiment model or pulse file.

FORMAT

\[ \text{status} = \text{MDS$OPEN}(\text{experiment}, \text{shot-number}) \]

RETURNS

VMS Usage: cond_value
  type: longword (unsigned)
  access: write only
  mechanism: by value

ARGUMENTS

experiment
  VMS Usage: char_string
  type: character string
  access: read only
  mechanism: by descriptor
  Argument specifying the experiment and optional sub-tree name. The
  experiment model and pulse files are located using the experiment name
  and shot number. When referencing a sub-tree, append the sub-tree name
  to the experiment name separating the experiment and sub-tree with a
  dollar character (i.e., experiment$sub-tree).

shot-number
  VMS Usage: longword
  type: longword
  access: read only
  mechanism: by reference
  Argument specifying the shot number. Two shot numbers are reserved for
  referencing the current shot number and the experiment model.
  • Shot 0 — current shot for experiment
  • Shot -1 — experiment model

DESCRIPTION

The MDS$OPEN routine will open an experiment model or pulse file. This
routine must be called before any calls to MDS$GET or MDS$PUT. A
stack of opened pulse/model files are maintained for your program unless
the files are explicitly closed via a call to MDS$CLOSE. If you need to
repeatedly switch back and forth between multiple pulse files, you can
reselect a previously opened file by calling MDS$OPEN again. If the file
is already open, MDS$OPEN simply sets the context of the subsequent
MDS$GET and MDS$PUT calls to point to the desired file.
### Data Access Routines

#### MDS$OPEN

<table>
<thead>
<tr>
<th>RETURN VALUES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE$ _NORMAL</td>
<td>Operation successfully completed</td>
</tr>
<tr>
<td>TREE$ _NOT_OPEN</td>
<td>No trees currently open</td>
</tr>
<tr>
<td>TREE$ _INVTREE</td>
<td>Internal memory pointers have been corrupted in your program.</td>
</tr>
<tr>
<td>RMS$ _xxxxxx</td>
<td>Any error returned from the VMS record management system.</td>
</tr>
</tbody>
</table>
MDS$PUT—Put data

The MDS$PUT routine writes data into a node of an experiment model or pulse file.

**FORMAT**

\[
\text{status} = \text{MDS$PUT}(\text{path-name}, \text{expression} [, \text{arg1}, \ldots, \text{argn}])
\]

**RETURNS**

<table>
<thead>
<tr>
<th>VMS Usage</th>
<th>cond_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type:</td>
<td>longword (unsigned)</td>
</tr>
<tr>
<td>access:</td>
<td>write only</td>
</tr>
<tr>
<td>mechanism:</td>
<td>by value</td>
</tr>
</tbody>
</table>

**ARGUMENTS**

**path-name**

<table>
<thead>
<tr>
<th>VMS Usage</th>
<th>path name</th>
</tr>
</thead>
<tbody>
<tr>
<td>type:</td>
<td>character string</td>
</tr>
<tr>
<td>access:</td>
<td>read only</td>
</tr>
<tr>
<td>mechanism:</td>
<td>by descriptor</td>
</tr>
</tbody>
</table>

Argument specifying the name of a node in the experiment model or pulse file. Path names can be either absolute or relative.

Absolute path names begin with a tag specifier, which has the format:

\[\text{\textbackslash{}[tree-name::]tag-name}\]

the tag-name can be any tag defined during model creation or the reserved "TOP" tag signifying the top of the tree. Absolute path names are not dependent on the current default node (with one exception: if the tree-name is not specified, the tag is looked up in the tag table of the tree or subtree of the current default node).

Relative path names do not begin with a tag specifier and the node name is resolved relative to the current default. If no leading punctuation is used, a leading ":" is assumed.

**expression**

<table>
<thead>
<tr>
<th>VMS Usage</th>
<th>char_string</th>
</tr>
</thead>
<tbody>
<tr>
<td>type:</td>
<td>character string</td>
</tr>
<tr>
<td>access:</td>
<td>read only</td>
</tr>
<tr>
<td>mechanism:</td>
<td>by descriptor</td>
</tr>
</tbody>
</table>

Argument specifying an MDSplus expression. The expression can be as simple as a node name or as complex as an arithmetic expression containing any number of node names, constants or argument placeholders. See the MDSplus Expression Evaluator manual for detailed syntax and functionality of MDSplus expressions.
Data Access Routines
MDS$PUT

)argn

VMS Usage: value

type: any

access: read only

mechanism: by descriptor

Argument specifying a replacement argument to be inserted in the expression in place of argument placeholders (i.e., $, $1, $2, $n). These optional arguments permit the user to insert the value of local variables in the evaluation of MDSplus expressions. Pass local variables to this routine by descriptor. There is one special exception to this passing mechanism which is in the use of data-id's returned by the MDS$GET routine. When using a data-id, you must pass the data-id by value.

DESCRIPTION

The MDS$PUT routine will write data into a node of an experiment model or pulse file. All types of data supported by MDSplus can be written using this routine. See examples below.

RETURN VALUES

<table>
<thead>
<tr>
<th>TREE$_NORMAL</th>
<th>Operation successfully completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE$_NOT_OPEN</td>
<td>No trees currently open</td>
</tr>
<tr>
<td>TREE$_INVTREE</td>
<td>Internal memory pointers have been corrupted in your program.</td>
</tr>
<tr>
<td>RMS$_xxxxxx</td>
<td>Any error returned from the VMS record management system.</td>
</tr>
</tbody>
</table>

EXAMPLES

The following examples illustrate the use of the MDS$PUT routine.

1

Real*4 CALIBRATION
CALIBRATION = 9876.5432
CALL MDS$OPEN('CMOD',-1) ! Open experiment model file
CALL MDS$PUT('\CMOD$POWER::TOR_CAL',
+ '$',%DESCR(CALIBRATION)) ! Put calibration value in file
CALL MDS$CLOSE ! Close experiment model file
END

In the above simple example, a floating point calibration constant is written into the experiment model file in the node "\CMOD$POWER::TOR_CAL". Note that since a local variable was written, the expression contains nothing but a single argument placeholder where the calibration variable is to be inserted.

2

Subroutine MY_PUT_IMAGES(IMAGES)
Byte IMAGES(256,256,10)
CALL MDS$PUT('\CMOD$XRAY::TOP.PINHOLE_CAM:XRAY_IMAGES'
+ '"',%DESCR(IMAGES)) ! Put an 3-D array into file
RETURN
END

The above example shows how a 3-D array can be inserted into a pulse file. Again, since a local variable was written the expression is a simple placeholder.
Subroutine MY_PUT_EXPRESSION
Integer*4 BEST_ONE

! Set to default node
CALL MDS$SET_DEFAULT('\CMOD\XRAY::TOP\PINHOLE\CAM')
CALL CHECK_MEASUREMENTS(BEST_ONE) ! User routine to get best value to use
CALL MDS$PUT('BEST_ANSWER', ! Put expression using the best gain and
  + 'XRAY\_IMAGES \* GAINS[$1] + OFFSETS[$1]', ! offset
  + %DESCR(BEST_ONE))
RETURN
END

The above example shows an expression being inserted into a pulse file. In this simplified example, a user routine is called to decide which constants in the pulse file will yield the best answer when the data in the 'XRAY\_IMAGES' node is multiplied by the n'th element of the vector in the 'GAINS' node and added to the n'th element of the vector in the 'OFFSETS' node. Note the use of the '$1' placeholder to re-use the first argument following the expression text.
MDS$SET_DEFAULT—Set Default Node

The MDS$SET_DEFAULT routine sets the default node location in the experiment model or pulse file. Subsequent path-relative path references (no leading backslash) are resolved relative to the current default.

**FORMAT**

```
status = MDS$SET_DEFAULT(path-name)
```

**RETURNS**

VMS Usage: `cond_value`
type: longword (unsigned)
access: write only
mechanism: by value

**ARGUMENTS**

`path-name`

VMS Usage: `path name`
type: character string
access: read only
mechanism: by descriptor

Argumet specifying the name of a node in the experiment model or pulse file. Path names can be either absolute or relative.

Absolute path names begin with a tag specifier, which has the format:

\ [tree-name::]tag-name

the tag-name can be any tag defined during model creation or the reserved "TOP" tag signifying the top of the tree. Absolute path names are not dependent on the current default node (with one exception: if the tree-name is not specified, the tag is looked up in the tag table of the tree or subtree of the current default node).

Relative path names do not begin with a tag specifier and the node name is resolved relative to the current default. If no leading punctuation is used, a leading ":" is assumed.

**DESCRIPTION**

The MDS$SET_DEFAULT routine sets the default node location in an opened model or pulse file. Subsequent path-relative path references are resolved relative to this default node. When a model or pulse file is first opened, the current default node is the top node of the model. Referencing nodes that are several levels deep in the tree structure from the top node can produce very long path names. Using the MDS$SET_DEFAULT routine, you can 'move' to the area of the tree where most of your node references are and then reference the nodes using much shorter names. See examples below.
RETURN VALUES

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE$_NORMAL</td>
<td>Operation successfully completed</td>
</tr>
<tr>
<td>TREE$_NOT_OPEN</td>
<td>No trees currently open</td>
</tr>
<tr>
<td>TREE$_INVTREE</td>
<td>Internal memory pointers have been corrupted in your program.</td>
</tr>
</tbody>
</table>

EXAMPLES

The following examples illustrate the convenience of using the MDS$SET_DEFAULT routine:

```
1
Real*4 CHAN1(1024)
Real*4 CHAN2(1024)
Real*4 CHAN3(1024)
Integer*
Integer*4 DLENGTH
CALL MDS$OPEN('CMOD',0)  ! Open the pulse file
CALL MDS$VALUE('CMOD::TOP.POWER.MAGNETS.TOROIDAL:CHANNEL_1',%DESCR(CHAN1),DLENGTH)
CALL MDS$VALUE('CMOD::TOP.POWER.MAGNETS.TOROIDAL:CHANNEL_2',%DESCR(CHAN2),DLENGTH)
CALL MDS$VALUE('CMOD::TOP.POWER.MAGNETS.TOROIDAL:CHANNEL_3',%DESCR(CHAN2),DLENGTH)
CALL MDS$CLOSE
```

The following example performs the same function but utilizes the MDS$SET_DEFAULT routine:

```
2
Real*4 CHAN1(1024)
Real*4 CHAN2(1024)
Real*4 CHAN3(1024)
Integer*
Integer*4 DLENGTH
CALL MDS$OPEN('CMOD',0)  ! Open the pulse file
CALL MDS$SET_DEFAULT('CMOD::TOP.POWER.MAGNETS.TOROIDAL')  ! Set default node
CALL MDS$VALUE('CHANNEL_1',%DESCR(CHAN1),DLENGTH)
CALL MDS$VALUE('CHANNEL_2',%DESCR(CHAN2),DLENGTH)
CALL MDS$VALUE('CHANNEL_3',%DESCR(CHAN2),DLENGTH)
CALL MDS$CLOSE
```

Note how the path names specified in the MDS$VALUE calls have shortened and the code becomes much readable.
MDS$VALUE—Get value

The MDS$VALUE routine returns the data specified by an expression in the users variable.

**FORMAT**

\[
\text{status} = \text{MDS$VALUE} ( \text{expression} [\text{, arg1, ..., argn}], \\
\text{user-variable, [data-size]} )
\]

**RETURNS**

VMS Usage: \text{cond\_value}  
\text{type: longword (unsigned)}  
\text{access: write only}  
\text{mechanism: by value}

**ARGUMENTS**

**expression**

VMS Usage: \text{char\_string}  
\text{type: character string}  
\text{access: read only}  
\text{mechanism: by descriptor}  
Argument specifying an MDSplus expression. The expression can be as simple as a node name or as complex as an arithmetic expression containing any number of node names, constants or argument placeholders. See the MDSplus Expression Evaluator manual for detailed syntax and functionality of MDSplus expressions.

**argn**

VMS Usage: \text{value}  
\text{type: any}  
\text{access: read only}  
\text{mechanism: by descriptor}  
Argument specifying a replacement argument to be inserted in the expression in place of argument placeholders (i.e., $\$, $\$, $\$, $\$, $\$, $\$). These optional arguments permit the user to insert the value of local variables in the evaluation of MDSplus expressions. Pass local variables to this routine by descriptor. There is one special exception to this passing mechanism which is in the use of data-id’s returned by the MDS$GET routine. When using a data-id, you must pass the data-id by value.

**user-variable**

VMS Usage: \text{data}  
\text{type: any}  
\text{access: write}  
\text{mechanism: by descriptor}  
The \text{user-variable} argument is the address of a descriptor of a variable to receive the data. An attempt will be made to convert the data to the data type of the variable receiving the data. If a conversion is impossible an error status will be returned. In FORTRAN, character variables are passed by descriptor by default. However, variables of all other data types (such as Real*4, Real*8, Integer*2, Integer*4) are passed by reference by
default. To pass these types of variables by descriptor use the %DESCR() built-in. The size of the user-variable does not necessarily need to match the size of the data. As much of the data as possible will be loaded into the user variable. If the user variable is larger than the data, it will be padded with zeros with the exception of character variables, which will be padded with blanks.

**data-size**

VMS Usage: longword

type: longword (signed)

access: write only

mechanism: by reference

The data-size argument is the address of a longword to receive the number of elements contained in the data obtained by evaluating the expression. If the user-variable argument is a character type variable, the data-size returned will represent the number of characters. Note the value returned in data-size may be larger, smaller or the same size as the user-variable argument.

Caution: If this argument is omitted, the comma after the user-variable and before the closing parenthesis is required.

**DESCRIPTION**

The MDS$VALUE routine will evaluate the expression specified and return the result in the user variable converted to the data type of the user variable. The user variable does not need to match in size or shape with the result. As much of the data as will fit into the user variable will be returned.

**RETURN VALUES**

<table>
<thead>
<tr>
<th>TREE$_NORMAL</th>
<th>Operation successfully completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE$_NOT_OPEN</td>
<td>No trees currently open</td>
</tr>
<tr>
<td>TREE$_INV_TREE</td>
<td>Internal memory pointers have been corrupted in your program</td>
</tr>
</tbody>
</table>

**EXAMPLES**

The following example the use of the MDS$VALUE routine:

```plaintext
Integer*2 IDATA(8192)
Integer*4 ILENGTH
Real*4 RDATA(8192)
Integer*4 RLENGTH
Real*4 COEF /2.1E-5/
Real*4 OFFSET /5.93/
Character*20 COMMENT
Integer*4 CLENGTH
CALL MDS$OPEN('CMOD',0)
CALL MDS$SET_DEFAULT('.POWER.MAGNETS')
CALL MDS$VALUE('TOROIDAL',%DESCR(IDATA),ILENGTH)
CALL MDS$VALUE('TOROIDAL \* $1 / $2',%DESCR(COEF),%DESCR(OFFSET),
   + %DESCR(RDATA),RLENGTH)
CALL MDS$VALUE('COMMENT',COMMENT,CLENGTH)
CALL MDS$CLOSE
END
```

In the above example, the data contained in the 'TOROIDAL' node
Data Access Routines

MDSS$VALUE

is returned as Integer*2 data in the variable IDATA. The result of multiplying all the elements of the data contained in the 'TOROIDAL' node by the values of COEF and OFFSET are returned as Real*4 data in the variable RDATA. The comment text contained in the 'COMMENT' node is returned in the COMMENT variable.
## Index

### D
Data-id • MDS$–5, MDS$–6, MDS$–11, MDS$–16

### E
Experiment • MDS$–3, MDS$–9
Experiment model • MDS$–3, MDS$–6, MDS$–9, MDS$–11, MDS$–14, MDS$–16
Expression • MDS$–6, MDS$–11, MDS$–16

### M
MDS$CLOSE • MDS$–3
MDS$FREE • MDS$–5
MDS$GET • MDS$–6
MDS$OPEN • MDS$–9
MDS$PUT • MDS$–11
MDS$SET_DEFAULT • MDS$–14
MDS$VALUE • MDS$–16

### P
Path name • MDS$–11, MDS$–14
Pulse file • MDS$–3, MDS$–6, MDS$–9, MDS$–11, MDS$–14, MDS$–16

### R
Routines
MDS$CLOSE • MDS$–3
MDS$FREE • MDS$–5
MDS$GET • MDS$–6
MDS$OPEN • MDS$–9
MDS$PUT • MDS$–11
MDS$SET_DEFAULT • MDS$–14
MDS$VALUE • MDS$–16

### S
Shot number • MDS$–3, MDS$–9