Accessing MDS with IDL
Reference Manual

February 1993

Software Version:
MDS 5.2
VAX/VMS Version 4.5 or greater

MDS Development
Massachusetts Institute of Technology
Cambridge, Massachusetts
February 1993

The information in this document is subject to change without notice and should not be construed as a commitment by the Massachusetts Institute of Technology. MIT assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No guarantee is made that the software is adequately or completely described in, or behaves in accordance with, this document.

Copyright ©1986 by Massachusetts Institute of Technology

All Rights Reserved.

Printed in U.S.A.

The following are trademarks of Digital Equipment Corporation:

CDD DECnet VAX
DATATRIEVE DECUS VAXcluster
DEC MicroVAX VAX Information Architecture
DEC/CMS MicroVMS VMS
DEC/MMS Rdb/VMS VT

IDL (Interactive Data Language) is copyrighted by Research Systems, Incorporated.

GRAPAC4 (Graphics Package 4) is copyrighted by Massachusetts Institute of Technology.

This document was prepared using TeX, the typesetting system developed by Donald E. Knuth at Stanford University. TeX is a registered trademark of the American Mathematical Society.
1 Introduction

IDL is a software system for the interactive analysis, reduction and display of scientific data. It is designed to help scientists to quickly and accurately analyze their data. IDL combines an immediate mode interactive compiler, a powerful set of array oriented operators and functions, with extensive graphic and image display capabilities into a truly interactive system.

Designed for scientists and engineers, IDL has a simple, easy to learn, structured syntax. It allows the user to concentrate on problem solving, rather than the mundane details of routine program development.

Your MDS system includes interface software so that IDL can reference MDS data directly from MDS databases. If you have IDL on your system you can now access MDS database items directly from within the IDL analysis environment and manipulate and display the data interactively. In most cases, it makes writing FORTRAN data analysis programs unnecessary.
Interface Initialization

Before you can access MDS within IDL you must first define a logical name to direct IDL to search the MDSIDL text library for the MDS procedures and functions. When a an unknown procedure or function is referenced in IDL, IDL searches text libraries for the existence of this procedure/function. It locates these libraries by finding the logical names IDL$LIBRARY, IDL$LIBRARY_1, IDL$LIBRARY_2, etc. When IDL is installed, there are one or two text libraries included with the IDL package and your system manager may have the logical names IDL$LIBRARY and IDL$LIBRARY_1 pointing to these files. These will generally be system logical names. To instruct IDL to also look for the MDS procedures and functions, your system manager can add a logical name, for example, IDL$LIBRARY_2 pointing to the file, SYS$LIBRARY:MDSIDL.TLB. This can be done at the same time the other IDL$LIBRARY logicals are defined, usually during system startup.

If your system manager has not defined these names, you may have to define them yourself in your process logical name table. IDL first searches your process logical name table for IDL$LIBRARY,IDL$LIBRARY_1 ... before searching for the same series of logicals in the system table. You could define the logical pointing to the MDS text library in your login command file as follows:

$ DEFINE IDL$LIBRARY SYS$LIBRARY:MDSIDL.TLB

Once you have properly included the MDS interface library in IDL's search path for finding procedures and functions, you will be able to access MDS within your IDL sessions. When you first enter IDL, you must enter the command MDS in response to the IDL prompt. This command will activate a shared image which contains all the interface software enabling IDL to make MDS database access calls on your behalf. After issuing the MDS command, you can invoke any of the MDS/IDL procedures or functions described in this manual. The MDS command need only be issued one time. For this reason, it is recommended that you include an MDS command in your IDL startup file. (See the IDL User's Guide provided with the IDL software for information on startup files.)
Procedure/Function Summary

The following procedures are available via the MDS interface software:

ATTACH            Attach to another process in job
CCL                Perform CCL commands
CSV                Perform CSV commands
MDS                Initialize MDS interface
PCL                Perform PCL commands
SET_DB             Select MDS database
SET_MISSING        Set missing value representation
WFEVENT            Wait for MDS event
WFSHOT             Wait for event and set default shot

The following functions are also available:

ACT_DATE           Return date of last item accessed
ACT_SHOT           Return shot of last item accessed
DATA               Access database item
GET_DATE           Get current date
GET_SHOT           Get current shot
SET_DATE           Set default date
SET_END            Set end index for item retrievals
SET_ERRORS         Set error handling for database accesses
SET_INC            Set increment for item retrievals
SET_SHOT           Set default shot number
SET_START          Set start index for item retrievals
SHOTS              Return list of shots that are on-line
SOURCE             Return source of last item accessed
UNITS              Return units of last item accessed
Sample Session

The following sample session demonstrates how one might analyze MDS data using IDL both in an interactive mode and through the use of IDL procedures. The first session will show a simple session where a single database item is analyzed:

$ IDL
VAX IDL, (C) 1983, RSI.
IDL>MD
IDL>SET_DB,'DUA0:[DEMO]DEMO.MDSDB'
IDL>XRAY = DATA('SOFT_XRAY_08')
IDL>PLT,XRAY
Sample Session

Since all the interesting data happened to be within the first 3000 or so elements of the vector, we could extract only this section for further analysis:

IDL> \texttt{XRAY = XRAY(0:3000)}
IDL> PLOT, XRAY
Sample Session

To avoid some of the jitter from the signal, we could use a boxcar average smoothing function to remove some of the noise:

IDL> PLOT, SMOOTH(XRAY, 15)
Sample Session

or we could use a median filter:

IDL> PLOT, MEDIAN(BYTSCL(XRAY), 35)
For the next example, an IDL procedure was created to draw a fairly complex display. The resulting display will contain plots of several signals and a surface plot of a two dimensional surface, created from several MDS database items. The following IDL procedure was created to perform this task:

```idl
PRO TEST, A
  ; Procedure TEST
  SET_DB, 'DUA0: [DEM0] MDS_DB'
  ERASE
  CURRENT_SHOT = GET_SHOT(0)
  SET_VIEWPORT, 0, 1, 0, 1
  SET_XY, 0, 1, 0, 1
  !FANCY = 1
  XYOUTS, .3, .95, '!Alcator-C Shot ' + STRTRIM(STRING(CURRENT_SHOT, 2), 2)
  !NOERAS = 1
  !FANCY = 0

  SET_PLOT, !DEVICE
  SET_XY
  ; Plot the current versus time
  !XTITLE = 'Time (sec)'
  !YTITLE = 'Current'
  SET_VIEWPORT, .1, .5, .65, .85
  X = SET_INC(10)
  X = SET_END(3000)
  !XTICKS = 0
  !YTICKS = 0
  !TYPE = 0
  PLOT, DATA('SH_TOKEN4_TM'), (2100 + (-1 * DATA('SH_TOKEN4_08'))) / 100 ; Use DATA function
  !NOERAS = -1
  !YTITLE = '
  !XTITLE = '
  X = SET_INC(20)
  X = SET_END(8000)
  !XTICKS = 1
  !YTICKS = 1
  !TYPE = 28

  SET_VIEWPORT, .7, .95, .82, .92
  PLOT, DATA('SH_TOKEN1_02') - 2048
  SET_VIEWPORT, .7, .95, .72, .82
  PLOT, DATA('SH_TOKEN1_05') - 2048
  SET_VIEWPORT, .7, .95, .62, .72
  PLOT, MEDIAN(BYTSCL(DATA('SH_TOKEN1_08') - 2048), 5)
  X = SET_INC(10)
  X = SET_END(3333)

  SET_VIEWPORT, .7, .95, .52, .62
  PLOT, DATA('SH_TOKEN2_03')
  SET_VIEWPORT, .7, .95, .42, .52
  PLOT, DATA('SH_TOKEN2_04')
  SET_VIEWPORT, .7, .95, .32, .42
  PLOT, DATA('SH_TOKEN2_11')
  SET_VIEWPORT, .7, .95, .22, .32
```

4-5 MDS/IDL
PLOT, DATA('SH_TOK2_12')
SET_VIEWPORT(7,.95,.12,.22)
PLOT, DATA('SH_TOK2_16')
SET_VIEWPORT(7,.95,.02,.12)
PLOT, DATA('SH_TOK2_20')
SET_VIEWPORT(0,.6,.0,.5)
OLDEND=SET_END(3000)
OLDINC=SET_INC(120)
NAME = 'SOFT_XRAY'
SIG = STRING(NAME,'_','-1',2A,12.2) ; Construct 2 dimensional
IGIN=DATA(SIG)
RAW=DATA(SIG) ; array from database items
LEN=SIZE(RAW)
LEN = LEN(1)
SURF = 10^-((RAW-2048)/409.6/2.5.0)
FOR I = 2,16 DO BEGIN
SIG = STRING(NAME,'_','-1',2A,12.2)
X = DATA(SIG)
RAW = [ [RAW], [X] ]
VOLTS = (X-2048)/409.6/2.5.0
ENDFOR
SURF = BYTSCAL(SURF) ; Byte scale the surface
SURF = MEDIAN(SURF,3) ; Use median smoothing
SURFACE,SURF,300,30,.8,-.25.
RETURN ; Draw the surface plot
END

This procedure when activated created the following display:
Command Descriptions
Command Descriptions

ACT_DATE

Return the date associated with the last MDS data item referenced.

**FORMAT**

date = ACT_DATE( dummy )

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

date

The return value of this IDL function representing the date associated with the last data item referenced. The value returned will be a string containing a date in the format `dd-mon-year`.

dummy

For IDL to recognize ACT_DATE as a function reference and not a variable reference you must include an argument to the function. Since nothing is done with this argument, it is suggested that you just provide a scalar constant, for example 0, for this argument.

**DESCRIPTION**

This function returns the 'actual date' value associated with the last MDS data item referenced via a DATA function call. The 'actual date' is usually the same as the current default date used prior to doing the DATA function. However, transforms and vectors stored in a transform file may apply to groups of shots possibly spanning multiple days. The 'actual date' in this instance would be the first day for which this item applies and therefore may differ from the date requested.

**EXAMPLE**

```
IDL> X = DATA("MY_SIGNAL")
IDL> DATE = ACT_DATE(0)
IDL> PRINT, DATE
13-FEB-1986
IDL>
```

This example shows the retrieval of an MDS data item, MY_SIGNAL, and the use of the ACT_DATE function to obtain the date of the data item.
**Command Descriptions**

**ACT_SHOT**

**ACT_SHOT**

Return the shot associated with the last MDS data item referenced.

**FORMAT**

```
shot = ACT_SHOT( dummy )
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

**SHOT**

The return value of this IDL function representing the shot associated with the last data item referenced. The value returned will be a 4 byte integer.

**dummy**

For IDL to recognize ACT_SHOT as a function reference and not a variable reference you must include an argument to the function. Since nothing is done with this argument, it is suggested that you just provide a scalar constant, for example 0, for this argument.

**DESCRIPTION**

This function returns the 'actual shot' value associated with the last MDS data item referenced via a DATA function call. The 'actual shot' is usually the same as the current default shot used prior to doing the DATA function. However, transforms and vectors stored in a transform file may apply to multiple shots. The 'actual shot' in this instance would be the first shot for which this item applies and therefore may differ from the shot number requested.

**EXAMPLE**

```
IDL> x = DATA("MY_SIGNAL")
IDL> shot = ACT_SHOT(0)
IDL> print, shot
1
IDL>
```

This example shows the retrieval of an MDS data item, *MY_SIGNAL*, and the use of the ACT_SHOT function to obtain the shot of the data item.
ATTACH

Enables you to switch control from your current process to another process in your job.

<table>
<thead>
<tr>
<th>FORMAT</th>
<th>ATTACH, process-name</th>
</tr>
</thead>
</table>

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

The name of the process you wish to attach to. This can be an IDL variable or a string constant. Process names can be up to 15 characters in length.

Note: This argument is case sensitive. If you use a quoted string and include lowercase letters, it will not match the process name if the process name is made up of all uppercase letters. If you omit the quotes on the process name argument, IDL will convert it to uppercase.
Command Descriptions

ATTACH

DESCRIPTION

The ATTACH procedure allows you to connect your input stream to another process. You can use ATTACH to change control from one subprocess to another subprocess or to the parent process. (No connection can be made to the current process, to a process that is not part of the current job, or to a process that does not exist.)

EXAMPLE

IDL> $
$ SET PROCESS/NAME=MY_SUBPROCESS
$ ATTACH TWF
IDL> ATTACH,'MY_SUBPROCESS'
%DCL-S-RETURNED, control returned to process MY_SUBPROCESS
$ LOGOUT
  Process MY_SUBPROCESS logged out at 19-SEP-1986 11:44:10.56
IDL>

This example shows a subprocess being created with the "$" command in IDL. The DCL "SET PROCESS/NAME" command is used to change the process name of the subprocess to "MY_SUBPROCESS". The DCL "ATTACH" command attaches back to the original parent process running IDL. Then from IDL, the ATTACH procedure is invoked to get control back to the subprocess. Once the subprocess is logged out, control returns to the parent process again. the use of the ACT_SHOT function to obtain the shot of the data item.
CCL

Perform CCL (CAMAC Command Language) command from IDL.

**FORMAT**

```
CCL , "[ccl-command]"
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

`ccl-command`

A valid CCL command. Enclose the command in single or double quotes. You can also use an IDL variable if the variable is of string type. If omitted, IDL will enter the CCL utility and will prompt for CCL commands. Exiting from CCL will return to IDL.
Command Descriptions
CCL

DESCRIPTION
The CCL command permits the execution of CCL commands from within the IDL environment. These commands are executed within the process context of IDL and no subprocess is created. (See the Camac Command Language manual for descriptions of the available commands).

Since the ccl-command parameter can be an IDL string variable, it is possible to construct CCL commands with IDL procedures and issue those commands directly from within IDL. If you are using an string for the command, it must be enclosed in quotes.

If the CCL procedure is used without any parameters, your process will enter CCL command mode and you will receive prompts for CCL commands. To return to IDL you must exit from CCL. While in CCL command mode you cannot issue IDL commands.

The first time you enter an MDS 'MDSCL' interpreter such as CCL or CSV, the initialization command file, if any, will be executed. For example, if the CCL procedure is invoked and it is the first MDS command interpreter procedure execution, the CCL utility will behave much as if you had issued the CCL command from DCL. That is, the utility will look for the presence of an initialization file called CCLINIT and invoke the commands in that initialization file. (If no logical name is present for CCLINIT the interpreter will search for SYS$LOGIN:CCLINIT.CCL). Subsequent CCL procedure invocations or any other MDS interpreter procedure invocations, for that matter, will NOT execute an initialization procedure.

EXAMPLES

IDL>CCL,"SET MODULE MY_DIGITIZER"
IDL>SETUP = 4096 * (CHANNELS - 1)
IDL>COMMAND = "PIO/FUNC=16/ADDRESS=0/DATA=" + STRTRIM(SETUP, 2)
IDL>CCL, COMMAND
IDL>CCL,"PIO/FUNC=9"

The example above issues a CCL command to establish the default CAMAC module for subsequent CCL commands. A setup word is computed within IDL and a CCL command is constructed using this value. The CCL command to load the setup word is the issued. Then one last CCL command is issued, in this case, starting a digitizer on its sweep and log mode.

IDL> CCL
CCL> PIO/FUNC=25
CCL> STOP/COUNT=100/FUNC=2
CCL> EXIT
IDL>

This second example show the CCL procedure invoked without any parameters. The CCL prompt is received and CCL commands are entered. After the
CCL "EXIT" command is issued, the program is returned to the IDL prompt.
**CSV**

Perform CSV (CAMAC SerVer Language) command from IDL.

| **FORMAT** | 
| --- | --- |
| CSV , "[csv-command]" | 

| **restrictions** | Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information. |

<table>
<thead>
<tr>
<th><strong>command parameter</strong></th>
<th>csv-command</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid CSV command. Enclose the command in single or double quotes. You can also use an IDL variable if the variable is of string type. If omitted, IDL will enter the CSV utility and will prompt for CSV commands. Exiting from CSV will return to IDL.</td>
<td></td>
</tr>
</tbody>
</table>
**DESCRIPTION**

The CSV command permits the execution of CSV commands from within the IDL environment. These commands are executed within the process context of IDL and no subprocess is created. (See the CAMAC SerVer Language manual for descriptions of the available commands).

Since the *CSV-command* parameter can be an IDL string variable, it is possible to construct CSV commands with IDL procedures and issue those commands directly from within IDL. If you are using a string for the command, it must be enclosed in quotes.

If the CSV procedure is used without any parameters, your process will enter CSV command mode and you will receive prompts for CSV commands. To return to IDL you must exit from CSV. While in CSV command mode you cannot issue IDL commands.

The first time you enter an MDS 'MDS DCL' interpreter such as CSV or CCL, the initialization command file, if any, will be executed. For example, if the CSV procedure is invoked and it is the first MDS command interpreter procedure execution, the CSV utility will behave much as if you had issued the CSV command from DCL. That is, the utility will look for the presence of an initialization file called CSVINIT and invoke the commands in that initialization file. (If no logical name is present for CSVINIT the interpreter will search for SYS$LOGIN:CSVINIT.CSV). Subsequent CSV procedure invocations or any other MDS interpreter procedure invocations, for that matter, will NOT execute an initialization procedure.

**EXAMPLES**

**IDL**

```
SET_DB, "MY_DATABASE"
CSV, "SHOW DEFAULT"
```

Database: MY_DATABASE  Date: 19-SEP-1986  Shot: 70

```
CSV
SET MDF MY_SCANTABLE
MODIFY/MDF MY_DIGITIZER/ON
EXIT
```

The example above invokes an IDL procedure to establish the MDS database. A CSV command shows the default date and shot for the MDS database.

```
CSV
```

This second example show the CSV procedure invoked without any parameters. The CSV prompt is received and CSV commands are entered. After the CSV "EXIT" command is issued, the program is returned to the IDL prompt.
DATA

Get data item from MDS database.

**FORMAT**

\[
\text{value} = \text{DATA}(\text{item-name})
\]

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL\$STARTUP file. See the MDS procedure for more information.

**command parameter**

**value**

The return value of this IDL function. If no errors occur, the DATA function will return a vector of floating point numbers representing the data stored in the MDS database for the specified database item. If an error occurs during this function indicating that the data was not found, the function will return a scalar value of -1. (See the SET_ERRORS function for more information on error handling during the execution of the DATA function).

**item-name**

The name of the MDS database item you are retrieving. MDS database items have names from 1 to 23 characters in length. The DATA function will convert the name you provide to uppercase prior to performing the database access. You can use an IDL string variable or a string constant for this argument.
**DESCRIPTION**

This function retrieves a database item from the MDS database. Prior to issuing a DATA function call, you must select an MDS database using the SET_DB procedure. (See the SET_DB procedure for more information.) You may also use the SET_SHOT and SET_DATE procedures to change your default date and shot to be used in the database access. (See the SET_SHOT and/or SET_DATE functions for more information.)

You can limit the amount of data returned using the SET_START, SET_END and/or the SET_INC functions. These functions will set the start, end, and increment values to be used when retrieving data. For example, setting start to 5, end to 100, and increment to 10 would return a subset of the specified item representing every tenth element beginning with element 5 and ending with element 100. When using the SET_START or SET_END functions, the first element is considered data element 1 (not 0 as is the case in IDL array indexing). To reset the end index to return all data, you must use the SET_END function specifying a 0 for the end index.

Once a database item has been successfully retrieved, you can use the ACT_DATE, ACT_SHOT, SOURCE, and UNITS functions to obtain ancillary information about the database item.

**EXAMPLE**

```idl
IDL> X = DATA("MY_SIGNAL")
IDL> DATE = ACT_DATE(0)
IDL> PRINT, DATE
13-FEB-1986
IDL> HELP, X
X , FLT, ARR, DIMS=( 8000)
```

This example shows the retrieval of an MDS data item, *MY_SIGNAL*, and the use of the ACT_DATE function to obtain the date of the data item. The IDL procedure, "HELP", reveals the data item returned is a floating point array with 8000 elements.
GET_DATE

Return the current default date used for MDS database accesses.

**FORMAT**

\[
\text{date} = \text{GET\_DATE}(\text{dummy})
\]

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

**date**

The return value of this IDL function representing the default date used in accessing MDS database items. The value returned will be a string containing a date in the format `dd-mon-year`.

**dummy**

For IDL to recognize GET\_DATE as a function reference and not a variable reference you must include an argument to the function. Since nothing is done with this argument, it is suggested that you just provide a scalar constant, for example 0, for this argument.

**DESCRIPTION**

This function returns the default date to be used when accessing MDS database items. When IDL is first invoked, the default date is set to use the "current" date, or date of last shot taken, for the database selected with a SET\_DB procedure call. The default date can be altered using the SET\_DATE function. (See the SET\_DATE function for more details.)

**EXAMPLE**

```idl
IDL> SET\_DB, MY\_DATABASE
IDL> OLD = SET\_DATE("LAST")
IDL> DATE = GET\_DATE(0)
IDL> PRINT, DATE
13\-FEB\-1986
```

This example shows the selecting of an MDS database, `MY\_DATABASE`, and the use of the GET\_DATE function to obtain the date of the "current" shot.
GET_SHOT

Return the current default shot used for MDS database accesses.

**FORMAT**

\[
\text{shot} = \text{GET\_SHOT}(\text{dummy})
\]

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL\$STARTUP file. See the MDS procedure for more information.

**command parameter**

*shot*

The return value of this IDL function representing the default shot used in accessing MDS database items. The value returned will be a four byte integer.

*dummy*

For IDL to recognize GET\_SHOT as a function reference and not a variable reference you must include an argument to the function. Since nothing is done with this argument, it is suggested that you just provide a scalar constant, for example 0, for this argument.

**DESCRIPTION**

This function returns the default shot to be used when accessing MDS database items. When IDL is first invoked, the default shot is set to use the “current” shot, the shot number of last shot taken, for the database selected with a SET\_DB procedure call. The default shot can be altered using the SET\_SHOT function. (See the SET\_SHOT function for more details.)

**EXAMPLE**

```
IDL> SET\_DB, MY\_DATABASE
IDL> OLD = SET\_SHOT(“-1”)  
IDL> SHOT = GET\_SHOT(0) 
IDL> PRINT, SHOT          70
IDL>
```

This example shows the selecting of an MDS database, *MY\_DATABASE*, and the use of the GET\_SHOT function to obtain the shot number of the “current” shot.
Command Descriptions

MDS

MDS

Enable MDS functions and procedures.

<table>
<thead>
<tr>
<th>FORMAT</th>
<th>MDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>It is necessary to invoke the MDS procedure, once and only once, to enable the remaining MDS functions and procedures. The MDS command activates the MDS interface shared images and defines several internal functions and procedures. It is recommended that you put an MDS procedure call in an IDL$STARTUP procedure. In this way, the MDS functions and procedures will be available for your use during the IDL session.</td>
</tr>
</tbody>
</table>

EXAMPLE

```
IDL> MDS
IDL> SET_DB, 'MY_DATABASE'
IDL>
```

In this example, the MDS procedure was invoked and then the MDS procedure SET_DB was invoked. If one attempted to issue the SET_DB command prior to the MDS command, and error would occur.
PCL

Perform PCL (Plot Command Language) command from IDL.

**FORMAT**

```
PCL,"[pcl-command]"
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

```
PCL-command
```

A valid PCL command. Enclose the command in single or double quotes. You can also use an IDL variable if the variable is of string type. If omitted, IDL will enter the PCL utility and will prompt for PCL commands. Exiting from PCL will return to IDL.
Command Descriptions

PCL

DESCRIPTION

The PCL command permits the execution of PCL commands from within the IDL environment. These commands are executed within the process context of IDL and no subprocess is created. (See the Plot Command Language manual for descriptions of the available commands).

Since the `pcl-command` parameter can be an IDL string variable, it is possible to construct PCL commands with IDL procedures and issue those commands directly from within IDL. If you are using a string for the command, it must be enclosed in quotes.

If the PCL procedure is used without any parameters, your process will enter PCL command mode and you will receive prompts for PCL commands. To return to IDL you must exit from PCL. While in PCL command mode you cannot issue IDL commands.

The first time you enter an MDS 'MDSDCL' interpreter such as PCL or CCL, the initialization command file, if any, will be executed. For example, if the PCL procedure is invoked and it is the first MDS command interpreter procedure execution, the PCL utility will behave much as if you had issued the PCL command from DCL. That is, the utility will look for the presence of an initialization file called PCLINIT and invoke the commands in that initialization file. (If no logical name is present for PCLINIT the interpreter will search for SYS$LOGIN:PCLINIT.PCL). Subsequent PCL procedure invocations or any other MDS interpreter procedure invocations, for that matter, will NOT execute an initialization procedure.

Note: Plotting from PCL is handled entirely separate from plotting within IDL. For example, any IDL commands which establish the output device and the type of that device have no effect on PCL plotting and vice-versa. Also, you cannot use PCL to draw pictures of IDL variables.
**EXAMPLES**

**IDL>** PCL,"DEVICE TT:/REGIS"
**IDL>** CHANNEL = CHANNEL + 1
**IDL>** PCL,"PICTURE"
**IDL>** COMMAND = "DRAW/AXIS TIMING CHANNEL_" + STRTRIM(CHANNEL,2)
**IDL>** PCL,COMMAND
**IDL>** PCL,"END/HOPAUSE"
**IDL>**

The example above issues a PCL command to establish the device and type for output. An IDL command computes a channel number and then the data for that channel is plotted using PCL. Note the PCL command is constructed as an IDL string variable.

**IDL>** PCL
PCL> DEFINE MY_MACRO
DEFMAC> PICTURE
DEFMAC> DRAW/AXIS TIMING CHANNEL_1
DEFMAC> END/HOPAUSE
DEFMAC>
PCL> EXIT
**IDL>**

This second example show the PCL procedure invoked without any parameters. The PCL prompt is received and PCL commands are entered. After the PCL "EXIT" command is issued, the program is returned to the IDL prompt.
## PUTMDS

Put IDL variable into MDS database.

### FORMAT

```plaintext
PUTMDS, item-name, units, idl-variable
```

### restrictions

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

### command parameter

#### item-name

The name of the MDS database item you are creating. MDS database items have names from 1 to 23 characters in length. The PUTMDS function will convert the name you provide to uppercase prior to performing the database access. You can use an IDL string variable or a string constant for this argument. The item-name can be prefixed with a MDS database level. If this level is omitted, the item will be placed in the 'P.' level.

#### units

The units of the MDS database item being created. You can use an IDL string variable or a string constant for this argument. MDS item units can be up to 8 characters in length.

#### idl-variable

An IDL variable whose value(s) will be stored in MDS. This variable or IDL expression can be a scalar value or an array. If it is a multi-dimensional array, the data is stored in MDS as a one dimensional array. Only variables of type byte, word, longword or floating point are permitted.
DESCRIPTION
This function stores a database item into the MDS database. Prior to issuing a PUTMDS function call you must select an MDS database using the SET_DB procedure. (See the SET_DB procedure for more information.) You may also use the SET_SSHOT and SET_DATE procedures to change your default date and shot to be used in the database access. (See the SET_SSHOT and/or SET_DATE functions for more information.) The database item will always be stored using the current default shot identification.

EXAMPLE

IDL> X = SIN(FINDGEN(100)/10.)
IDL> SET_DB,'MY_DATABASE'
IDL> TEMP = SET_SSHOT(400)
IDL> PUTMDS,'P.MY_SINE_WAVE','volts',X
IDL>

The example shows the creation of an IDL variable, X. An MDS database is selected and the default shot is specified. The PUTMDS procedure is used to write to the MDS database and create an item called *MY_SINE_WAVE* in the *P* level of the database.
**SETEVENT**

Generate an MDS event.

**FORMAT**

```
SETEVENT, event-name
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

```
event-name
```

Specifies the name of the event to generate. Event names can be up to 25 characters in length. You can specify a string constant or the name of an IDL string variable.

**DESCRIPTION**

The SETEVENT procedure allows you to generate an MDS event from within IDL. Other processes can wait for this event, by name, to occur before proceeding. Inter-process synchronization can be achieved using events. Essentially all MDS supported utilities support the generation and/or waiting for MDS events. These events can be detected in processes running on different nodes of a VAX cluster as well as processes running on the same CPU. Events are recognized either on a system wide basis or between processes of the same group depending on the way your MDS system is installed.

**EXAMPLE**

```
IDL> SETEVENT, 'MY_EVENT'
```

In the example above, the event ,’MY_EVENT’, is issued.
### SET_DATE

Sets the current default date used for MDS database accesses.

#### FORMAT

```plaintext
old-date = SET_DATE( new-date )
```

#### restrictions

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

#### command parameter

- **old-date**
  The return value of this IDL function representing the previous default date used in accessing MDS database items. The value returned will be a string containing a date in the format `dd-mon-year` or one of the keywords 'LAST' or 'TODAY'.

- **new-date**
  Specifies the new default date to be used when accessing data in an MDS database. The argument can be a string constant or an IDL string variable. The date should be in the format `dd-mon-year` or one of the keywords 'LAST' or 'TODAY'. Setting the date to 'LAST' will make the date of the last or current shot of the MDS database selected the default date.
Command Descriptions

SET_DATE

DESCRIPTION
This function sets the default date to be used when accessing MDS database items. When IDL is first invoked, the default date is set to use the “current” (LAST) date, or date of last shot taken, for the database selected with a SET_DB procedure call. The default date resolved to an actual date can be obtained using the GET_DATE function. (See the GET_DATE function for more details.) For ‘SHOT ONLY’ databases the SET_DATE function will have no effect since shots in this database type are accessed by shot number only.

EXAMPLE

IDL> SET_DB, MY_DATABASE
IDL> OLD = SET_DATE(‘22-SEP-1986’)
IDL> PRINT, OLD
LAST
IDL> OLD = SET_DATE(‘TODAY’)
IDL> PRINT, OLD
22-SEP-1986
IDL> OLD = SET_DATE(‘TODAY’)
IDL> PRINT, OLD
TODAY
IDL>
IDL> DATE = GET_DATE(0)
IDL> PRINT, DATE
23-SEP-1986
IDL>
This example shows the selecting of an MDS database, MY_DATABASE, and the use of the SET_DATE function to set the default date to first 22-SEP-1986 then to ‘TODAY’. Note the difference in the value returned from SET_DATE when the previous default was ‘TODAY’ and the value returned from GET_DATE.
**SET_DB**

Select MDS database. This procedure must be invoked prior to any MDS database access functions are used. The SET_DB procedure identifies the MDS database from which data is to be retrieved.

**FORMAT**

```
SET_DB, database-name
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

`database-name`

Specifies a file containing the description of an MDS database. This may be a logical name or the file specification itself. If providing a file specification you can omit the directory, name, type, and/or version fields. The defaults for these fields are `'[000000]MDS.DATABASE;0'`.

**EXAMPLE**

```
IDL> SET_DB, 'ALCATOR_DATABASE'
IDL> PRINT, GET_DATE(), GET_SHOT()
22-SEP-1986 76

IDL> SET_DB, 'DUA0: [TWF] MY OWN_DATABASE'
IDL> PRINT, GET_DATE(), GET_SHOT()
22-SEP-1986 1
```

This example shows two SET_DB procedure invocations. The first part selects the `'ALCATOR_DATABASE'` database and prints out the default date and shot for that database. The second part selects a private database and prints out the default date and shot for it.
# SET_END

Sets the element index of the last value returned from the `DATA` function.

## FORMAT

| old-end = SET_END( new-end ) |

## restrictions

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

## command parameter

- **old-end**
  The return value of this IDL function representing the previous default end index used in accessing MDS database items. The value returned will be a four-byte integer.

- **new-end**
  Specifies the new default end index to be used when accessing data in an MDS database. The value must be either an integer constant or an integer IDL variable name. Specify a value of zero to reset the end index to the last element of the item retrieved.
**DESCRIPTION**

When using the DATA function to retrieve data from MDS, you can request a subset of the data to be returned by specifying a start, end and/or increment to be used for computing element indexes. For example, with start = 5, end = 100 and increment = 10, the DATA function would return the 5th, 15th, 25th, ... and the 95th elements of the data item requests (provided the item had over 95 elements). To set these index ranges you can use the SET_START, SET_END and/or the SET_INCREMENT functions prior to performing a DATA function call. Once these indexing parameters are set they remain in effect until they are changed. Initially the start and increment values are set to 1 and the end is set to 0 (dynamically uses the index of the last element of the data). With these settings all elements are retrieved.

**EXAMPLE**

```
IDL> SET_DB, MY_DATABASE
IDL> X = DATA('MY_CHANNEL_01')
IDL> HELP, X
   X , FLT, ARR, DIMS=( 8000)
IDL> OLD_START = SET_START(1000)
IDL> OLD_END = SET_START(7000)
IDL> OLD_INC = SET_INC(100)
IDL> X = DATA('MY_CHANNEL_01')
IDL> HELP, X
   X , FLT, ARR, DIMS=(  61)
IDL>
```

In this example a SET_DB procedure is used to select an MDS database. The DATA function is then used to retrieve data for the item called 'MY_CHANNEL_01'. Using the IDL HELP procedure, we find that there are 8000 elements in returned array 'X'. However if the start, end, and increments are set to 1000, 7000 and 100 a small subset of the data (elements 1000, 1100, 1200, ..., 7000) are returned creating an array of 61 elements.
SET_ERRORS

Establish error handling for MDS database accesses.

**FORMAT**

old-setting = SET_ERRORS( new-setting )

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

**old-setting**

The return value of this IDL function representing the previous setting of the error handling. A string will be returned, either ‘NONE’, ‘BRIEF’, ‘FULL’, or ‘ABORT’. When IDL is first invoked the default setting is ‘FULL’.

**new-setting**

A quoted string or an IDL string variable containing one of the following keywords: ‘NONE’, ‘BRIEF’, ‘FULL’, or ‘ABORT’. Any other keyword will produce an error message.
**Command Descriptions**

**SET_ERRORS**

**DESCRIPTION**
There are four basic ways that IDL can deal with errors while performing MDS database accesses ranging in severity from no error messages to the abortion of IDL procedures.

The four classes available are:
- ‘NONE’ No message, just return a scalar -1 value
- ‘BRIEF’ Brief message, return a scalar -1 value
- ‘NONE’ Full message, return a scalar -1 value
- ‘NONE’ Brief message and abort the current procedure

**EXAMPLES**

```
IDL> OLD = SET_ERRORS("FULL")
IDL> X = DATA("MY_SIGNAL")
%TDB$-E-RNF, Error reading "MY_SIGNAL" for shot 13-FEB-1986 \#99
  -RMS-E-RNF, record not found
IDL> PRINT, X
   -1

This first example shows the error handling when set to “FULL”. This is the initial setting when IDL is first invoked.

IDL> OLD = SET_ERRORS("NOTE")
IDL> X = DATA("MY_SIGNAL")
IDL> PRINT, X
   -1

The next example shows the error handling when set to “NONE”. Note that no error message was issued but the function returned a -1 value.

IDL> OLD = SET_ERRORS("BRIEF")
IDL> X = DATA("MY_SIGNAL")
%TDB$-E-RNF, Error reading "MY_SIGNAL" for shot 13-FEB-1986 \#99
IDL> PRINT, X
   -1

This example shows the error handling when set to “BRIEF”. A one line message is issued in this mode.

IDL> OLD = SET_ERRORS("ABORT")
IDL> X = DATA("MY_SIGNAL")
*** %TDB$-E-RNF, Error reading "MY_SIGNAL" for shot 13-FEB-1986 \#99. Error detected at:
LINE 0, ROUTINE $$MAIN$$.
IDL> PRINT, X
*** Variable is not defined, Name= X. Error detected at:
LINE 0, ROUTINE $$MAIN$$.
```
Command Descriptions
SET_ERRORS

IDL>

This example shows the error handling when set to "ABORT". Note the procedure aborted and no value was loaded into the X variable.
SET_INC

Sets the element increment used when retrieving MDS data.

**FORMAT**

```
old-inc = SET_INC( new-inc )
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

- **old-inc**
  The return value of this IDL function representing the previous default increment used in accessing MDS database items. The value returned will be a four byte integer.

- **new-inc**
  Specifies the new increment to be used when accessing data in an MDS database. The value must be either a integer constant or an integer IDL variable name. Specify a value of one to reset the increment to include all elements.
Command Descriptions
SET_INC

DESCRIPTION
When using the DATA function to retrieve data from MDS, you can request a subset of the data to be returned by specifying a start, end and/or increment to be used for computing element indexes. For example, with start = 5, end = 100 and increment = 10, the DATA function would return the 5th, 15th, 25th, ... and the 95th elements of the data item requests (provided the item had over 95 elements). To set these index ranges you can use the SET_START, SET_END and/or the SET_INCREMENT functions prior to performing a DATA function call. Once these indexing parameters are set they remain in effect until they are changed. Initially the start and increment values are set to 1 and the end is set to 0 (dynamically uses the index of the last element of the data). With these settings all elements are retrieved.

EXAMPLE

IDL> SET_DB, MY_DATABASE
IDL> X = DATA('MY_CHANNEL_01')
IDL> HELP, X
X, FLT, ARR, DIMS=(8000)
IDL> OLD_START = SET_START(1000)
IDL> OLD_END = SET_START(7000)
IDL> OLD_INC = SET_INC(100)
IDL> X = DATA('MY_CHANNEL_01')
IDL> HELP, X
X, FLT, ARR, DIMS=(61)
IDL>

In this example a SET_DB procedure is used to select an MDS database. The DATA function is then used to retrieve data for the item called 'MY_CHANNEL_01'. Using the IDL HELP procedure, we find that there are 8000 elements in returned array 'X'. However if the start, end, and increments are set to 1000, 7000 and 100 a small subset of the data (elements 1000, 1100, 1200, ... , 7000) are returned creating an array of 61 elements.
SET_MISSING

Specifies a value to be used for missing values in data returned from MDS database accesses.

**FORMAT**

SET_MISSING, missing-value

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

missing-value

Specifies a floating point value to be substituted for any missing values found in the data retrieved from MDS. You can use a floating point constant or an IDL variable.

**DESCRIPTION**

In certain instances, MDS data accesses may return data arrays with one or more elements have missing values. Certain MDS Transforms for example may return missing values as the result of invalid operations such as division by zero. Before these arrays are returned as IDL variables, all elements which are 'missing' are replaced by a floating point value which you can specify using the SET_MISSING procedure.

The default value used to represent missing is -1.7014117E+38.

**Note:** The missing value is only used within context of a MDS data retrieval. Invalid operations, division by zero etc., that occur during arithmetic operations within IDL do not generate missing values.

**EXAMPLE**

IDL> MISSING = 0.0
IDL> SET_MISSING,MISSING
IDL> X=DATA('MY_SIGNAL')
IDL> PLOT,X(WHERE(X NE MISSING))
IDL>

In this example, missing values are set to 0.0. After the data is retrieved, all elements of the array are plotted except for missing values.
SET_SHOT

Sets the current default shot used for MDS database accesses.

**FORMAT**

```
old-shot = SET_SHOT( new-shot )
```

**command parameter**

- **old-shot**
  The return value of this IDL function representing the previous default shot used in accessing MDS database items. The value returned will be a four byte integer.

- **new-shot**
  Specifies the new default shot to be used when accessing data in an MDS database. The argument can be an integer constant or an IDL variable. Specify the value, -1, to set the default shot to the 'last' or 'current' shot for the database.

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.
DESCRIPTION  This function sets the default shot to be used when accessing MDS database items. When IDL is first invoked, the default shot is set to use the "current" shot, or shot number of the last shot taken, for the database selected with a SET_DB procedure call. The default shot, resolved to an actual shot number, can be obtained using the GET_SHOT function. (See the GET_SHOT function for more details.)

EXAMPLE

IDL> SET_DB, MY_DATABASE
IDL> OLD = SET_SHOT(78)
IDL> PRINT, OLD
   -1
IDL> OLD = SET_SHOT(-1)
IDL> PRINT, OLD
   78
IDL> OLD = SET_SHOT(-1)
IDL> PRINT, OLD
   -1
IDL> SHOT = GET_SHOT(0)
IDL> PRINT, SHOT
   96

This example shows the selecting of an MDS database, *MY_DATABASE*, and the use of the SET_SHOT function to set the default SHOT first to 78 then to -1. Note the difference in the value returned from SET_SHOT when the previous default was -1 and the value returned from GET_SHOT.
SET_START

Sets the element index of the first value returned from the DATA function.

```
old-start = SET_START( new-start )
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL\$STARTUP file. See the MDS procedure for more information.

**command parameter**

- **old-start**
  The return value of this IDL function representing the previous default start index used in accessing MDS database items. The value returned will be a four byte integer.

- **new-start**
  Specifies the new default start index to be used when accessing data in an MDS database. The value must be either an integer constant or an integer IDL variable name. Specify a value of one to reset the start index to the first element of the item retrieved.
Command Descriptions

SET_START

**DESCRIPTION**

When using the DATA function to retrieve data from MDS, you can request a subset of the data to be returned by specifying a start, end and/or increment to be used for computing element indexes. For example, with start = 5, end = 100 and increment = 10, the DATA function would return the 5th, 15th, 25th, ... and 95th elements of the data item requests (provided the item had over 95 elements). To set these index ranges you can use the SET_START, SET_END and/or the SET_INCREMENT functions prior to performing a DATA function call. Once these indexing parameters are set they remain in effect until they are changed. Initially the start and increment values are set to 1 and the end is set to 0 (dynamically uses the index of the last element of the data). With these settings all elements are retrieved.

**EXAMPLE**

```
IDL> SET_DB, MY_DATABASE
IDL> X = DATA('MY_CHANNEL_01')
IDL> HELP, X
X , FLT, ARR, DIMS=( 8000)
IDL> OLD_START = SET_START(1000)
IDL> OLD_END = SET_START(7000)
IDL> OLD_INC = SET_INC(100)
IDL> X = DATA('MY_CHANNEL_01')
IDL> HELP, X
X , FLT, ARR, DIMS=( 61)
IDL>
```

In this example a SET_DB procedure is used to select an MDS database. The DATA function is then used to retrieve data for the item called 'MY_CHANNEL_01'. Using the IDL HELP procedure, we find that there are 8000 elements in returned array 'X'. However if the start, end, and increments are set to 1000, 7000 and 100 a small subset of the data (elements 1000, 1100, 1200, ..., 7000) are returned creating an array of 61 elements.
Command Descriptions

SHOTS

SHOTS

Return a list of shots for which data files have been found on-line.

**FORMAT**

\[ \text{shot-list} = \text{SHOTS( date[,prefix] )} \]

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

- **shot-list**
  The return value of this IDL function will be an array of longwords if any data files are found; otherwise, it will return a scalar value of -1. The array will contain one or more shot numbers corresponding to the data files found.

- **date**
  Specifies a date for which data files are to be located. The SHOTS function will return the list of shot numbers for which corresponding files have been found for this date. For ‘SHOT ONLY’ databases, the date field is ignored and the SHOTS function will return shot numbers for all data files found on-line.

  This argument can be a string constant or an IDL string variable. Specify dates in the format *dd-mon-year*.

- **[,prefix]**
  Optionally, specifies a database level prefix. If omitted, it will default to 'S'. Use a single character string constant or an IDL variable for the prefix.
**DESCRIPTION**

The SHOTS function returns an array of shot numbers for which data files can be located for the specified date. The mechanism for finding data files and their corresponding shots numbers is as follows.

First, a wild card file specification is constructed by substituting the date character elements into the appropriate locations in the file mask defined for the specified level (‘S’ prefix level is the default level). The character locations which would normally be associated with shot number characters are replaced with %’s, a wild card character. For example if the file mask for the specified level was “DUA0:[SHOTS.fghi|SHOT_ab_cde_fghi.qrs” and the date specified was “1-JAN-1986”, then the resulting wild card file specification would be “DUA0:[SHOTS.1986|SHOT_01_JAN.1986.%%%”.

Next, all the files matching this wild card specification are located. For each file found, a shot number is computed by back translating the resulting file specification based on the file mask. As each file is found, a corresponding shot number is appended to the array of shot numbers to be returned.

**EXAMPLE**

```idl
IDL> SET_DB, MY_DATABASE
IDL> SHOTS = SHOTS('22-SEP-1986')
IDL> PRINT, SHOTS
     1,  20,  99
IDL> $DIRECTORY DUA0:[SHOTS.1986]SHOT_22_SEP_1986.*
Total of 3 files.
IDL>
```

This example shows the selecting of an MDS database, *MY_DATABASE*, and the use of the SHOTS function to obtain a list of shot numbers on-line for the 22nd of September. The SHOTS function found 3 shot files and returned a 3 element vector containing the shot numbers.
SOURCE

Return the source associated with the last MDS data item referenced.

FORMAT

```
source = SOURCE( dummy )
```

restrictions

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

command parameter

- **source**: The return value of this IDL function representing the source associated with the last data item referenced. The value returned will be a character string.
- **dummy**: For IDL to recognize SOURCE as a function reference and not a variable reference you must include an argument to the function. Since nothing is done with this argument, it is suggested that you just provide a scalar constant, for example 0, for this argument.

DESCRIPTION

This function returns the source associated with the last MDS data item referenced via a DATA function call.

The source associated with an MDS database item is an identification of the program, module or user which put the item in the database. For example, the source may be the name of the module which acquired the data and stored it in the database or it may be the user which defined a transformation on the data.

EXAMPLE

```
IDL> X = DATA('SH_XAP_01')
IDL> Y = SOURCE(0)
IDL> PRINT, Y
SH_XAP
IDL>
```

This example shows the retrieval of an MDS data item, *SH_XAP_01*, and the use of the SOURCE function to obtain the source of the data item. This particular signal is the first channel of a CAMAC digitizer call ‘SH_XAP’.
UNITS

Return the units associated with the last MDS data item referenced.

**FORMAT**

```
units = UNITS( dummy )
```

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

- **units**
  The return value of this IDL function representing the units associated with the last data item referenced. The value returned will be a character string.

- **dummy**
  For IDL to recognize UNITS as a function reference and not a variable reference you must include an argument to the function. Since nothing is done with this argument, it is suggested that you just provide a scalar constant, for example 0, for this argument.

**DESCRIPTION**

This function returns the units of the data returned from the last DATA function call.

**EXAMPLE**

```
IDL> X = DATA('SH_XAP_01')
IDL> Y = UNITS(0)
IDL> PRINT,Y counts

IDL>
```

This example shows the retrieval of an MDS data item, \texttt{SH\_XAP\_01}, and the use of the UNITS function to obtain the UNITS of the data item. This particular signal is the first channel of a CAMAC digitizer call \texttt{SH\_XAP} and the units of the data returned are ADC counts.
Command Descriptions

WFEVENT

WFEVENT

The WFEVENT command waits for an MDS event to occur.

FORMAT

WFEVENT, event-name

restrictions

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

command parameter

event-name

Specifies the name of the event to wait for. Names are converted to uppercase. The event-name can be a string constant or and IDL string variable from 1 to 25 characters in length.

DESCRIPTION

The WFEVENT command will wait until the specified MDS event occurs before continuing. MDS events are either system wide or group wide depending on the way your MDS system was installed. These events provide a mechanism for synchronizing processes on one CPU or any node in a VAX cluster. MDS events do not propagate to other nodes linked via DECNET.

EXAMPLE

IDL> WFEVENT, MY_DIGITIZER
IDL> PLOT, DATA(MY_SIGNAL)

The preceding example waits for the event MY_DIGITIZER and then plots the MDS database item called MY_SIGNAL.
The WFSHOT procedure waits for an MDS event to occur and sets the default date and shot to represent the current date and shot of the database when the event was issued. Every time the event occurs subsequent to the WFSHOT invocation, the current shot number is placed in a to-be-processed queue. When the next WFSHOT procedure is invoked (must be same event), if the to-be-processed queue is empty, the process will wait for the event to occur. If the to-be-processed queue is not empty, the WFSHOT procedure will return immediately with the default shot set to the value in the queue. The WFSHOT procedure provides a mechanism of synchronizing with the added feature of allowing the IDL session to fall behind and then catch up later if necessary.

**FORMAT**

WFSHOT, event-name

**restrictions**

Before this procedure/function can be issued in IDL, the MDS procedure must be invoked. The MDS procedure, which makes all MDS procedures and functions available to the IDL session, should be invoked once prior to attempting any other MDS operation. It is recommended that an MDS procedure call be placed in your IDL$STARTUP file. See the MDS procedure for more information.

**command parameter**

**event-name**

Specifies the name of the event to wait for. Names are converted to uppercase. The event-name can be a string constant or and IDL string variable from 1 to 25 characters in length.
Command Descriptions
WFSHOT

DESCRIPTION

The WFSHOT procedure establishes an MDS event to use for synchronizing with a MDS shot cycle. Once the procedure is invoked, each time the event occurs the current default date/shot for the current MDS database will be placed in a to-be-processed queue. Each time the WFSHOT procedure is called specifying the same event-name the to-be-processed queue is checked. If there is something in the queue, the MDS default date and shot is set to the shot number in the queue and that shot is removed from the queue. If there is nothing in the queue, the WFSHOT procedure waits till the event occurs.

In this manner, it is possible to create IDL procedures which repetitively wait for shots to occur and then process the data. With the WFSHOT procedure, no shots can be skipped even though the IDL procedure may fall behind one or more shots.

Note: There can only be one active WFSHOT event-name. Whenever a WFSHOT procedure is invoked, the event-name is checked against that of the previous invocation. If it is not the same event, the to-be-processed queue is emptied and recognition of the previous event is discontinued.

If a valid shot-id is present in the lock value block associated with the MDS event, that shot-id is used instead of the ‘current’ shot for the database.

EXAMPLE

IDL> WHILE 1 DO BEGIN
IDL> WFSHOT, MY_DIGITIZER
IDL> PLOT, DATA(MY_SIGNAL)
IDL> ENDWHILE

The preceding example, repetitively, waits for the event MY_DIGITIZER and then plots the MDS database item called MY_SIGNAL. If another shot occurs while it is plotting, it won’t miss the shot.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>Section 1</td>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>Section 2</td>
<td>Interface Initialization</td>
<td>2-1</td>
</tr>
<tr>
<td>Section 3</td>
<td>Procedure/Function Summary</td>
<td>3-1</td>
</tr>
<tr>
<td>Section 4</td>
<td>Sample Session</td>
<td>4-1</td>
</tr>
</tbody>
</table>
### Section 5  Command Descriptions

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT_DATE</td>
<td>5-2</td>
</tr>
<tr>
<td>ACT_SHOT</td>
<td>5-3</td>
</tr>
<tr>
<td>ATTACH</td>
<td>5-4</td>
</tr>
<tr>
<td>CCL</td>
<td>5-6</td>
</tr>
<tr>
<td>CSV</td>
<td>5-9</td>
</tr>
<tr>
<td>DATA</td>
<td>5-11</td>
</tr>
<tr>
<td>GET_DATE</td>
<td>5-13</td>
</tr>
<tr>
<td>GET_SHOT</td>
<td>5-14</td>
</tr>
<tr>
<td>MDS</td>
<td>5-15</td>
</tr>
<tr>
<td>PCL</td>
<td>5-16</td>
</tr>
<tr>
<td>PUTMDS</td>
<td>5-19</td>
</tr>
<tr>
<td>SETEVENT</td>
<td>5-21</td>
</tr>
<tr>
<td>SET_DATE</td>
<td>5-22</td>
</tr>
<tr>
<td>SET_DB</td>
<td>5-24</td>
</tr>
<tr>
<td>SET_END</td>
<td>5-25</td>
</tr>
<tr>
<td>SET_ERRORS</td>
<td>5-27</td>
</tr>
<tr>
<td>SET_INC</td>
<td>5-30</td>
</tr>
<tr>
<td>SET_MISSING</td>
<td>5-32</td>
</tr>
<tr>
<td>SET_SHOT</td>
<td>5-33</td>
</tr>
<tr>
<td>SET_START</td>
<td>5-35</td>
</tr>
<tr>
<td>SHOTS</td>
<td>5-37</td>
</tr>
<tr>
<td>SOURCE</td>
<td>5-39</td>
</tr>
<tr>
<td>UNITS</td>
<td>5-40</td>
</tr>
<tr>
<td>WFEVENT</td>
<td>5-41</td>
</tr>
<tr>
<td>WFSHOT</td>
<td>5-42</td>
</tr>
</tbody>
</table>
Preface

Intended Audience

This manual is intended for users familiar with the IDL (Interactive Data Language) software package from Research Systems, Incorporated. It describes how to access MDS data from within IDL but does not document the functionality of the IDL package. For more information on IDL see the IDL User's Guide available from RSI.

Structure of This Document

This document describes the use of several functions and procedures provided with the MDS base system for accessing data stored in MDS databases from within the IDL software package. The following is a brief description of the topics covered in this manual:

- Section 1 Introduction, the introduction provides a general overview of IDL and the interface to MDS.

- Section 2 Interface Initialization, this section describes the steps necessary to make the MDS procedures available to your IDL session.

- Section 3 Procedure/Function Summary, this section describes the various categories of procedures and functions available for accessing MDS data.
Preface

- Section 4 Sample Session, this section provides a sample session using IDL, demonstrating how MDS data can be accessed and analyzed.

- Section 5 Procedure/Function Descriptions, this section provides detailed descriptions of the procedures and functions available for accessing MDS data.