User's Guide for SDDS-compliant EPICS Toolkit Version 1.5

L. Emery, M. Borland, H. Shang, R. Soliday
Advanced Photon Source

May 7, 2008

The SDDS-compliant EPICS toolkit is a set of software applications for the collection or writing of data in Experimental Physics and Industrial Control System (EPICS) database records. Though most of the applications essentially do rather simple operations, the combination of these and others from the SDDS postprocessing toolkit allow arbitrarily complicated analysis of data and control of the accelerators at the Advanced Photon Source. These tools are general and can be applied to devices other than accelerators under control of EPICS.

The EPICS tools presented here read and store data to SDDS-protocol files. SDDS (Self-Describing Data Set)[1] refers to a particular implementation of a self-describing file protocol used at APS. Self-describing means that the data is refered to and accessed by name. Thus, a user doesn't need to know, say, in which column a piece of tabular data is located. An ASCII header contains information about the file's data structure, i.e. definitions of structure elements such as columns (tabular data) and parameters (single values).

Initially adopted for complex physics simulation programs, it was clear that the SDDS file protocol would excel in data-collecting software as well. Typically, an EPICS tool would write EPICS data to an SDDS file with each readback written to a column of name corresponding to the EPICS database record name. Single value data that describe the experimental conditions might be written to the file as parameters. Once collected, the EPICS data can be further analyzed and plotted with any of the SDDS tools described in [1]. One can regard the EPICS tools as the layer between the EPICS control system and more functional analyzing tools and scripts, with SDDS protocol files as an intermediary.

Following conventional usage, EPICS database records will be refered to as "process variables" or PVs in this manual.

1 Manual Pages Overview

1.1 EPICS Toolkit Programs by Category

1.1.1 Configuration Save and Restore

- `burtrb` (2.1) — Reads values of process variables and writes them to a snapshot file. Example application: saving the configuration of an accelerator for later use.
- `burtwb` (2.2) — Reads values from a snapshot file and writes them to process variables. Example application: restore an earlier configuration of an accelerator.
- `sddscasr` (2.7) — a new configuration save and restore program with many unique features.

- `sddscaramp` (2.6) — Performs ramping of process variables between the present state and the states in one or more SDDS files.
- `toggle` (2.4) — Alternates between two sets of process variable values stored in snapshot files at a regular interval. Example application: Alternate between two configurations of an accelerator for debugging accelerator performance.

1.1.2 Data Collecting

- `sddssnapshot` (2.18) — Reads values of process variables and writes them to a snapshot file. Example application: Taking succesive sets of data.
- `sddsmonitor` (2.15) — Reads values of process variables and writes them to a file at regular intervals. Example application: archiving of system performance, investigation of glitches.
- `sddsglitchlogger` (2.12) — Reads values of process variables and writes them to a file or multifile at regular intervals when certain trigger occurs. Example application: archiving of system performance, inverstigation of transition-based, glitch-based and alarmed-based triggers.
- `sddsvmonitor` (2.22) — Reads values of process variables and writes them to a file at regular intervals. The list of readback process variables is constructed by multiplying a rootname list and a suffix list. Example application: archiving of system performance, investigation of glitches.
- `sddswmonitor` (2.23) — Reads values of waveform and scalar process variables and writes them to a file at regular intervals. Example application: archiving of system performance, investigation of glitches.
- `sddsstatmon` (2.19) — Reads values of process variables, calculates statistics and writes them to a file at regular intervals. Example application: archiving of system performance.
- `sddssynchlog` (2.20) — Reads values of process variables synchronously. Example application: Analyze for correlations between various systems.
- `sddsexperiment` (2.9) — Varies process variables and measures process variables, with optional statistical analysis. Example application: investigation of unforeseen physical dependences, measure corrector-bpm response matrices.
- `sddsvexperiment` (2.21) — Varies process variables and measures process variables, with optional statistical analysis. The list of readback process variables is constructed by multiplying a rootname list and a suffix list. Example application: investigation of unforeseen physical dependences, measure corrector-bpm response matrices.
- `sddsalarmlog` (2.5) — Collects alarm status of a list of process variables. Example applications: Logging alarms for a particular accelerator system over a period of many runs in order to compile a history of faults statistics.
- `sddslogger` (2.13) — Reads values of process variables and writes them to a file at regular intervals. Similar to sddsmonitor but it generates less network traffic.
- `sddslogonchange` (2.14) — Records only the values of process variables that change. This reduces the output file size if process variables do not change often.

1.1.3 Control

- `sddscontrollaw` (2.8) — Reads a matrix for a control law-type equation and regulates the values of a list of readback process variables by applying corrections to another list of process variables. Example application: removing slow drifts in accelerators (such as the energy of a linac beam), trajectory and orbit correction.

- `knobconfig` (2.3) — Assigns knobs to devices according to the input file. Example application: Manually tweak a trajectory, storage ring tunes and chromaticity.

- `squishPVs` (2.24) — Minimizes the absolute values of a set of readback process variables by varying setpoint process variables which have a physical influence on the readback process variables. Example application: Reducing the size of the trajectory in a beamline without knowledge of the response matrix between bpms and correctors.

- `sddsoptimize` (2.16) —Minimizes or maximizes the RMS of a set of readback process variables by automatically varying setpoint process variables (or knobs composed of setpoint PVs), which have a physical influence on the readback process variables, through simplex or 1dscan method.

- `sddspvtest` (2.17) —tests the process variable given by the inputfile are out-of-range or not and sets the number of out-of-range process variables to a control PV.

1.2 Toolkit Program Usage Conventions

(This text in this section is borrowed from [1].)

In order to make the multitude of Toolkit programs easier to use, the developers have attempted to use consistent commandline argument styles. The Toolkit programs all require at least one commandline argument. Therefore, if a program is executed without commandline arguments, it is assumed that the user is asking for help. In this case, a help message is printed that shows syntax and (usually) describes the meaning of the switches. In general, program usage is of the following form:

`programName` *fileNames switches*.

Probably the simplest example would be

`sddsquery` *fileName*,

which would invoke `sddsquery` to describe the contents of an SDDS file. A slightly more complicated example would be

`sddsquery` *fileName* `-columnList`,

which invokes `sddsquery` to list just names of columns in a file.

Programs assume that any commandline argument beginning with a minus sign ('-') is an option; all others are assumed to be filenames. Note that case is ignored in commandline switches. The specific meaning of a filename is dictated by its order on the commandline. For example, if two filenames are given, the first would commonly be an input file while the second would commonly be an output file.

In some cases, a command with a single filename implies replacement of the existing file. For example,

`sddsconvert` *fileName* `-binary`

would replace the named file with a binary version of the same data. This command is completely equivalent to

`sddsconvert -binary` *fileName*
That is, unlike many UNIX commands, the position of filenames relative to options is irrelevant.

One might also wish to make a new file, rather than replacing the existing file. This could be done by
`sddsconvert -binary` *fileName fileName2*
Note that while the option may appear anywhere on the commandline, the order of the filenames is crucial to telling the program what to do.

In following manual pages and in the program-generated help text, program usage is described using the following conventions:

- The first token on the commandline is the name of the program.
- Items in square-brackets (`[]`) are optional. Items not in square brackets are required.
- Items in curly-brackets ({}) represent a list of choices. The choices are separated by a `|` character, as in
  { *choice1* `|` *choice2* `|` *choice3* }
- Items in italics are descriptions of arguments or data that must be supplied by the user. These items are not typed literally as shown.
- Items in normal print are typed as shown, with optional abbreviation. These are usually switch keywords or qualifiers. Any unique abbreviation is acceptable.

In addition to using files, most toolkit programs also take input from pipes, which obviates the need for temporary files in many cases. For those programs that support pipes, one can employ the `-pipe` option. This option provides a good example of what options look like. For example, one could do the following to test binary-ascii conversion:
`sddsconvert -binary -pipe=out` *fileName* `| sddsconvert -ascii -pipe=in` *fileName1*
The `-pipe=out` option to `sddsconvert` tells it to deliver its output to a pipe; it still expects a filename for input. Similarly, the `-pipe=in` option to `sddsquery` tells it to accept input from a pipe.

The `-pipe` switch may be given in one of five forms: `-pipe`, `-pipe=input,output`, `-pipe=output,input`, `-pipe=input`, `-pipe=output` . The first three forms are equivalent. In a usage message, these forms would be summarized as `-pipe[=input][,output]`. One could also use abbreviations like `-pipe=i`, `-pipe=i,o`, etc. For convenience in the manual, the data stream from or to a pipe will often be referred to by the name of the file for which it substitutes. Note that you may not deliver more than one file on the same pipe.

2 Manual Pages

2.1 burtrb

- **description:** `burtrb` reads values of process variables and writes them to a file. An input file lists the process variables to be read. The output file (also called a snapshot file) can be used by program `burtwb` to restore the process variables.

- **example:** The state of the APS storage ring is saved in the snapshot file SR.snp using the request file SR.req:

  ```
  burtrb -f SR.req -o SR.snp
  ```

  where the contents of the file `SR.req` are

  ```
  SDDS1
  &column
   name = ControlType,  type = string, &end
  &column
   name = ControlName,  type = string, &end
  &data
   mode = "ascii", no_row_counts=1 &end
  pv      S1A:Q1:CurrentAO
  pv      S1A:Q2:CurrentAO
  ...
  ```

  Note that the header contain the minimal amount of information. There may be situations where more columns need to be defined, as described elswhere in this manual.

- **synopsis:**

  ```
  usage: burtrb -f req1 {req2 ...} {-l logfile} {-o outfile} {-d} {-v}
  {-c ... comments ...} {-k keyword1 ... keywordn}
  {-r retry count} {-sdds or -nosdds} {-Dname{=def}}
  {-Ipathname}

  where

    -f req1 {req2 ...} - Request filenames. This is the only switch
     that is not optional.  You must specify at least one request
     file.

    -l logfile - Log filename. The name of the file where all logging
     messages (e.g. error messages, reports of process variables
     that were not found) go.  The default is stderr.

    -o outfile - Snapshot filename.  The name of the file where the
     snapshot information goes.  The default is stdout.

    -d - Debug.  Save the files created by processing the request
     files with the C preprocessor.  The default is to delete these
  ```

```
 files.

-v - Verbose.  This increases the amount of information displayed
 in the logfile.

-c ... comments ... - Comments.  Adds comments to the header of
 the snapshot file.

-k keyword1 ... keywordn - Keywords.  Adds keywords to the header
 of the snapshot file.

-r retry count - Number of additional attempts to wait for
 connections.  The program will attempt to find all the process
 variables.  If it is unsuccessful, it will try this many more
 times to establish connections.  The default value is 0.

-sdds or -nosdds - SDDS/non-SDDS snapshot file.  Explicitly
 specifying that the generated snapshot file will be
 SDDS/non-SDDS compliant.  The default is to adopt the SDDS
 type from the input(s).  If there is a heterogeneous set of
 inputs (some SDDS and some non-SDDS), the default is to produce
 and SDDS compliant snapshot file.
```

- **files:**
  - **input file:**

    The input file is an SDDS file with at least two columns:

    * `ControlName` — Required string column for the process variable or device names.
    * `ControlType` — Required string column for the control name type. For a process variable name use "pv"; for a device name use "dev".

    The optional columns are:

    * `BackupMsg` — String column for the device read message if the `ControlType` value is "dev". This column is transfered to the output file. If this column is absent then the default read message is "read", and the column `BackupMsg` is created in the output file.
    * `RestoreMsg` — String column for the device set message if the `ControlType` value is "dev". This column is transfered to the output file. If this column is absent then the default read message is "read", and the column `RestoreMsg` is created in the output file.
  - **output file:**

    The output file contains the same columns as the input file plus additional ones:

    * `ValueString` — String column containing the readback value as a ASCII string.
    * `Lineage` — String column containing the composite device name. In the case that a composite device was given as a control name in the input file, `burtrb` will expand the composite device in its atomic devices, and write one row per atomic device in

the output file with the same `Lineage` value for each. If the control name in the input file was a process variable or an atomic device then the lineage is the character "-". (A quirk of program `burtrb` is the use of the character "-" for the equivalent of the empty string. A preferred way to specify a empty string would be simply "", but this hasn't been implemented in `burtrb`.)

    * `BackupMsg` — String column. If not specified in the input file, the default device read message is written to this column.
    * `RestoreMsg` — String column. If not specified in the input file, the default device write message is written to this column.

  Defined parameters in the output file are:

    * `TimeStamp` — String giving the date.
    * `BurtComment` — String given at the comment command line option.

- **switches:**
  - `-f file1 [file2] ...` — Request files.
  - `-o outfile` — Output file. If option is not present then stdout is the default.
  - `-c comment` — A comment string to be added to the output file.
- **see also:**
  - `burtwb` (2.2)
- **author: N. Karonis, ANL**
- **manual page: L. Emery, ANL**

2.2 burtwb

- **description:** `burtwb` reads values from a snapshot file, then write them to process variables. This is the restore counterpart of the backup program `burtrb`.
- **example:** The state of the APS storage ring is restored by writing the content of the snapshot file SR.snp to the appropriate process variables:

  ```
  burtwb -f SR.snp
  ```

  where the contents of the file `SR.snp` are

  ```
  SDDS1
  &description &end
  &parameter
   name = SnapType,  type = string, &end
  &column
   name = ControlType,  type = string, &end
  &column
   name = ControlName,  type = string, &end
  &column
   name = Lineage,  type = string, &end
  &column
   name = Count,  type = long, &end
  &column
   name = ValueString,  type = string, &end
  &data
   mode = ascii, no_row_counts=1 &end
  ! page number 1
  Absolute
  pv S1A:Q1:CurrentAO - 1 300.0
  pv S1A:Q2:CurrentAO - 1 400.0
  ```

- **synopsis:**

  ```
   usage: burtwb -f snap1 {snap2 ...} {-l logfile} {-o outfile}
                  {-c ... comments ...} {-k keyword1 ... keywordn}
                  {-d} {-v} {-p dep1 ... depn} {-r retry count} {-add} {-replace}
                  {-sdds or -nosdds}

  where

          -f snap1 {snap2 ...} - Snapshot filenames.  This is the only
                  switch that is not optional.  You must specify at least one
                  snapshot file.

          -l logfile - Log filename. The name of the file where all logging
                  messages (e.g. error messages, reports of process variables
                  that were not found) go.  The default is stderr.
  ```

```
-o outfile - Snapshot filename.  If any of the snapshot files
        read only notify values, this file is created and those values
        are placed there.  If none of the snapshot files have read only
        notify values, then no file is created.  The default is stdout.

-c ... comments ... - Comments.  Adds comments to the header of
        the snapshot file.

-k keyword1 ... keywordn - Keywords.  Adds keywords to the header
        of the snapshot file.

-d - Debug.  Save the files created by processing the dependency
        files with the C preprocessor.  The default is to delete these
        files.

-v - Verbose.  This increases the amount of information displayed
        in the logfile.

-p dep1 ... depn - Dependency filenames.  The names of the
        dependency files containing predicates to be evaluated before
        writing the values from the snapshot files.

-r retry count - Number of additional attempts to wait for
        connections.  The program will attempt to find all the process
        variables.  If it is unsuccessful, it will try this many more
        times to establish connections.  The default value is 0.

-add - Absolute snapshots written as adds.  All the absolute
        snapshots, i.e., those taken directly off IOCs, will be written
        as additions to the values found on the IOCs.  The default is
        to write the absolute snapshots as replacement values on the
        IOCs.

-replace - Relative snapshots written as replacements.  All the
        relative snapshots, i.e., those generated by adding or
        subtracting two snapshots, will be written to prelace the
        values on the IOCs.  The default is to write the relative
        snapshots as additions to the values on the IOCs.

-sdds or -nosdds - SDDS/non-SDDS snapshot file.  Explicitly
        specifying that the generated snapshot file will be
        SDDS/non-SDDS compliant.  The default is to adopt the SDDS
        type from the input(s).  If there is a heterogeneous set of
        inputs (some SDDS and some non-SDDS), the default is to produce
        and SDDS compliant snapshot file.
```

- **files:**

- **input file:**

  The input file is an SDDS file with at least three columns:

  * `ControlName` — Required string column for the process variable or device name.
  * `ControlType` — Required string column for the control name type. For a process variable name use "pv"; for a device name use "dev".
  * `Lineage` — Required string column for the lineage of a device name. The values aren't important for an input file. A value of `-` can be used (signifies a blank string to BURT programs).
  * `Count` — Required long column. Needs to have a value of 1 for PVs and devices that are single-valued, which is the case here.
  * `ValueString` — Required string column containing the value to be restored as a character string.

  and one parameter:

  * `SnapType` — Required string parameter describing the snapshot type. Valid values are `Absolute` and `Relative`.

  The optional columns are:

  * `RestoreMsg` — String column for the device set message if the `ControlType` value is "dev". If this column is absent then the default restore message is "set".

- **switches:**
  - `-f snap1 [snap1] ...` — Snapshot files.
  - `-add` — Adds the `ValueString` values to the process variables.
- **see also:**
  - `burtrb` (2.1)
- **author: N. Karonis, ANL**
- **manual page: L. Emery, ANL**

2.3 knobconfig

- **description:** `knobconfig` assigns knobs to devices according to the input file. Knob assignments are active only during program execution.
- **example:** Knobs are assigned for tune adjustment of the APS storage ring with this command:

  ```
  knobconfig SR_tunes.cokn
  ```

  where file `SR_tunes.cokn` contains weights for the various quadrupole magnets necessary for independent tune adjustments. Part of the first data set of the file `SR_tunes.cokn` is shown below:

  ```
  SDDS1
  &description
          text="Knob Assignment Specification"
          contents="knob assignment specification"
  &end
  &parameter name=ControlName, type=string  &end
  &parameter name=KnobDescription, type=string  &end
  &parameter name=ControlType,  type=string &end
  &parameter name=ControlUnits,  type=string &end
  &parameter name=Gain type=double &end
  &column name=ControlName type=string &end
  &column name=Weight type=double &end
  &data   mode=ascii, &end
  S:NUx:fine              ! ControlName
  Triplet knob for x tune (0.0005/click)  ! KnobDescription
  dev                     ! ControlType
  ?                       ! ControlUnits
  0.5                     ! Gain
  240                     ! number of rows
  S1A:Q1  1.488923268326000e-03
  S1A:Q2  5.958927747443000e-03
  S1A:Q3  4.242561180246000e-03
  S1B:Q3  4.242561180246000e-03
  S1B:Q2  5.958927747443000e-03
  S1B:Q1  1.488923268326000e-03
  S2A:Q1  1.488923268326000e-03
  S2A:Q2  5.958927747443000e-03
  ...
  ```

- **synopsis:**

  ```
  usage: knobconfig spec-file
  purpose: Configures knobs.
  Assigns knobs to devices and process variables
  according to spec-file. Assignments are active
  ```

```
only during program execution. In general, it
is best not to run this utility in background.
Sample spec-file. Note that ControlMessage parm is optional:
SDDS1
&description
        text="Knob Assignment Specification"
        contents="knob assignment specification"
&end
&parameter name=ControlName
           type=string
&end
&parameter name=ControlType
           type=string
&end
&parameter name=ControlMsg
           type=string
&end
&parameter name=ControlUnits
           type=string
&end
&parameter name=Gain
           type=double
&end
&column name=ControlName
        type=string
&end
&column name=Weight
        type=double
&end
&data   mode=ascii
        no_row_counts=1
&end
COMPDEVICENAME
dev
set
tune
0.01
P:Q1  1.0000
P:Q2  1.1134

PTB:Q3
dev
set
Amps
0.1

PTB:Q4:CurrentAO
pv
```

```
-
Amps
0.1
```

- **files:**
  - **input file:**
    Each data set in the input file defines the weights of the devices controlled by one knob. The required parameters are:
    * `ControlName` — String. The name of the composite device.
    * `ControlType` — String. Descibes the type of control name. The only valid value is "`dev`".
    * `Gain` — Double. Factor by which to multiply the `Weight` column values of each constituent of the composite device.

    An optional parameter is
    * `ControlMsg` — String. The device message to be used. The default value is "set".

    The parameters `KnobDesciption` and `Units` may be implemented in future versions of `knobconfig`. For now, they may be defined as a data preparation aid. The required columns are:
    * `ControlName` — String column of names of atomic devices.
    * `Weight` — Double column of relative weights of each atomic device. Multiplied with the `Gain`, these are the values by which each device setpoints will be changed by one tick of the knob.
- **switches:** None.
- **author: C. Saunders, ANL**

2.4 toggle

- **description:** `toggle` alternates between two sets of process variable values stored in snapshot files and/or process variables specified on the command line. Time intervals and the values of the process variables upon termination can be specified.
- **example:** Two states of the APS storage ring injection magnets are restored for 10 seconds each by writing values of process variables listed in the snapshot files SRin.snp1 and SRin.snp2:

  ```
  toggle SRin.snp1 SRin.snp2 interval=10,10
  ```

  In this case the magnets return returns to the initial state upon termination.
- **synopsis:**

  ```
  usage: toggle [snapshotfile1 [snapshotfile2]] [-controlname=PVname,value1[,value2] ...]
         [-interval=interval1[,interval2]] [-cycles=number] [-finalset={0|1|2}]
         [-prompt] [-verbose] [-warning]
  ```
- **files:**
  - **input files:**
    The input files are valid snapshot files as described in `burtwb` (2.2). At least three colums must be defined:
    * `ControlName` — Required column. String column for the process variable or device name.
    * `ControlType` — Required column. String column for the control name type. For a process variable name use “pv”; for a device name use “dev”.
    * `ValueString` — String column containing the value to be restored as a character string.

    Optional columns are:
    * `RestoreMsg` — Optional column. String column for the device set message if the `ControlType` value is “dev”. If this column is absent then the default read message is “set”.

    Both files must have the same set of PV names.
- **switches:**
  - `-controlname=<PVname>,<value1>[,<value2>]` — Optional. Specified a PV to be alternated. If only one value is given, then the second value is taken to be pre-existing. These values are synchronized with the snapshot files if they are present.
  - `-interval=<interval1>[,<interval2>]` — Optional. `<interval1>` and `<interval2>` are the durations of PV value sets 1 and 2, respectively. If `<interval2>` isn't present, then `<interval2>`=`<interval1>`. If this option isn't present, then the default interval is 1 second each.
  - `-cycles=<number>` — Optional number of cycles. Default is 1.
  - `-finalset=`{`0|1|2`} — Optional. Specifies which PV value set to apply at normal termination. 0,1, and 2 mean set of pre-existing values, first set and second set respectively. During abnormal termination, the PVs are returned to their pre-existing values.

  - `-prompt` — Optionally toggles values only on a `<CR>` key press.
  - `-verbose` — Optionally prints extra information.
  - `-warning` — Optionally prints warning messages.

- **see also:**
  - `burtwb` (2.2)

- **author: L. Emery, ANL**

2.5 sddsalarmlog

- **description:** `sddsalarmlog` logs changes of alarm status for process variables named in an input file.
- **example:** The status of the positron accumulator ring process variables are logged for a one day duration.

  ```
  sddsalarmlog PAR.alog  PAR.alog.sdds -timeDuration=1,day
  ```

  where the file `PAR.alog` is the input file containing the names of the process variables, and the file `PAR.alog.out` is the output file.
- **synopsis:**

  ```
  usage: sddsalarmlog <input> <output>
  -timeDuration=<realValue>[,<time-units>]
  [{-append[=recover] | -eraseFile | -generations[=digits=<integer>][,delimiter=<string>} |
   -dailyFiles[=verbose]]
  [-pendEventTime=<seconds>] [-durations] [-explicit[=only]] [-verbose]
  [-comment=<parameterName>,<text>]
  ```

- **files:**
  - **input file:**
    The input file is an SDDS file with the required string column `ControlName` which contains the names of the process variables whose status is to be monitored.
  - **output file:**
    The output file contains information on the process variable alarm status change, such as the time of occurence, the alarm designation, the severity of the alarm, and the row location of the previous alarm status change.

    In order to save disk space `sddsalarmlog` logs integer codes (indexes) instead of the actual string values for the control name, the alarm status and the alarm severity. The integer codes are indices into one of three string arrays written to the output file. The logging information can be recovered using SDDS tool sddsderef.

    As an option, and for a direct interpretation of the output file, the control name, alarm status and alarm severity can be written explicitly as string columns with or without the integer codes. However this option uses up a lot more disk space.

    In either case the following columns are defined:
    * `PreviousRow` — Long column of row numbers. For each process variables alarm status change, the row location of the previous alarm status change is written.
    * `TimeOfDay` — Float column of system time in units of hours. The time does not wrap around at 24 hours.

    By default these columns are defined (except when the option `-explicit=only` is specified):
    * `ControlNameIndex` — Long column indicating the process variable whose alarm status changed. The value of this data is the index into the string array `ControlNameString`, which is the list of all process variables monitored.

* AlarmStatusIndex — Short column indicating the alarm status. The value of this data is the index into the string array AlarmStatusString, which is the list of all possible alarm status values.
* AlarmSeverityIndex — Short column indicating the alarm status. The value of this data is the index into the string array AlarmSeverityString, which is the list of all possible alarm severity values.

These columns are created by the option -explicit:

* ControlName — String column for the process variable whose alarm status just changed.
* AlarmStatus — String column for the alarm status.
* AlarmSeverity — String column for the alarm severity.

This column is created with option -durations:

* Duration — String column for the duration of the previous alarm state.

These arrays are created by default except when the option -explicit=only is specified:

* ControlNameString — String colunm of all process variables to be monitored.
* AlarmStatusString — String colunm of all possible alarm status values.
* AlarmSeverityString — String colunm of all possible alarm severity values.

These parameters are defined:

* InputFile — String parameter for the name of the input file.
* TimeStamp — String parameter for time stamp for file.
* PageTimeStamp — String parameter for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The PageTimeStamp value for the new page is the creation date of the new page. The TimeStamp value for the new page is the creation date of the very first page.
* StartTime — Double parameter for start time from the C time call cast to type double.
* YearStartTime — Double parameter for start time of present year from the C time call cast to type double.
* StartYear — Short parameter for the year when the file was started.
* StartJulianDay — Short parameter for the day when the file was started.
* StartMonth — Short parameter for the month when the file was started.
* StartDayOfMonth — Short parameter for the day of month when the file was started.
* StartHour — Short parameter for the hour when the file was started.

- **switches:**
  - -timeDuration=<realValue>[,<time-units>] — Specifies time duration for logging. The default time units are seconds; one may also specify days, hours, or minutes.
  - -append[=recover] —Specifies appending to the file ¡output¿ if it exists already. If the recover qualifier is given, recovery of a corrupted file is attempted using sddsconvert, at the risk of loss of some of the data already in the file.
  - -eraseFile — If the output file already exists, then it will be overwritten by sddsalarmlog.

- -generations[=digits=<integer>][,delimiter=<string>] — The output is sent to the file <SDDSoutputfile>-<N>, where <N> is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating <N>, so that the first generation number is 0001.
- -pendEventTime=<seconds> — Specifies the CA pend event time, in seconds. The default is 10 .
- -durations — Specifies including state duration in output.
- -connectTimeout — Specifies maximum time in seconds to wait for a connection before issuing an error message. 60 is the default.
- -explicit[=only] — Specifies that explicit columns with control name, alarm status, and alarm severity strings be output in addition to the integer codes. If the "only" qualifier is given, the integer codes are omitted from the output.
- -verbose — Prints out a message when data is taken.
- -precision={single|double} — Selects teh data type for the statistics columns.
- -updateInterval=<integer> — Number of sample sets between each output file update. The default is 1.
- -ezcaTiming[=<timeout>,<retries>] — Sets EZCA timeout and retry parameters.
- -noezca — Obsolete.
- -dailyFiles – The output is sent to the file ¡SDDSoutputfile¿-YYYY-JJJ-MMDD.¡N¿, where YYYY=year, JJJ=Julian day, and MMDD=month+day. A new file is started after the midnight.
- -oncaerror={usezero|skiprow|exit} — Selects action taken when a channel access error occurs. The default is using zero (usezero) for the value of the process variable with the channel access error, and resuming execution. The second option (skiprow) is to

• **see also:**

• **author: M. Borland, ANL**

2.6 sddscaramp

- **description:** `sddscaramp` performs ramping of process variables between the present state and the states in one or more SDDS files.

- **example:** The following example shows how one would use `sddscaramp` to ramp 50% of the way to a new steering configuration, using 10 steps and pausing 1 second between steps:

  ```
  sddscaramp -rampTo=steering.snap,steps=10,pause=1,percentage=50
  ```

  The file `steering.snap` contains corrector magnet power supply settings, such as might be saved to an SDDS file using the `burtrb` program.

- **synopsis:**

  ```
  sddscaramp
    -rampTo=<filename>,steps=<number>,pause=<seconds>[,percentage=<value>]
    [-rampTo ...] [-dataColumn=<name>] [-verbose]
  ```

- **files:**

  The input files are SDDS files. There must be a string column named `ControlName`, `Device`, or `DeviceName` that contains the process variable names. There must also be a string column named `ValueString`, a numerical column named `Value`, or a column of either type with the name specified by the `-dataColumn` option; this column contains the final value for the corresponding process variable.

  When data is supplied in a string column, `sddscaramp` needs a way to determine if the data value is actually a number rather than a literal string value (e.g., an enumerated value). The optional `IsNumerical` column can be used for this purpose. If supplied, this column should contain character values `y` or `n`, indicating that each PV (respectively) does or does not have numerical values. If the `IsNumerical` column does not exist or is not of character type, then `sddscaramp` uses an internal algorithm to decide whether the data for each PV is numerical or not. This may fail in the case of enumerated values that contain numbers, resulting in incorrectly restored values. For reliable results, the use of `IsNumerical` with string data is required. If the data is in a numerical column to begin with, of course, there is no ambiguity.

- **switches:**

  - `-rampTo=`*`filename`*`,steps=`*`number`*`,pause=`*`seconds`*`,[percentage=`*`value`*`]` — Specifies a file to ramp to, the number of steps in the ramp, and the time to wait between sending setpoints for each step. Optionally, one may specify ramping only part of the way to the configuration in the file. If several `-rampTo` options are given, `sddscaramp` ramps to each of them in the order given.
  - `-dataColumn=`*`name`* — Specifies the name of a string column in the input files that contains the PV values. By default, the program uses `ValueString`.
  - `-verbose` — If given, informational text is printed out as the ramping proceeds.

- **see also:**

  - `burtrb` (2.1)

  - `burtwb` (2.2)

- **author: M. Borland (ANL)**
- **programmers: R. Soliday, M. Borland (ANL)**

2.7 sddscasr

- **description:** `sddscasr` is an alternative version of casave and carestore. sddscasr is more efficient and has more features than casave/carestore. It can also replace burtrb/burtwb for saving and restoring configurations.
- **example:** Save a snapshot of APS storage ring:

  ```
  sddscasr SR.req SR.snapshot -save -pendIOTime=100
  ```

  Restore a snapshot:

  ```
  sddscasr snapshot -restore -pendIOTime=100
  ```

  save snapshot with daemon mode, the output file in following command is out1-¡current date and time¿. Whenever the value of casavePV (oag:casave) is changed to 1, a new saving starts and data is written to a new output file with rootname of out1. (var1 is the input file contains PVs to be read)

  ```
  sddscasr var1 out1 -runControlPV=string=oag:ControlLawRC
  -runControlDesc=string=test -daemon -daily -save -pidFile=pidFile
   -casavePV=oag:casave -logFile=logFile &
  ```

- **synopsis:**

  ```
  usage: sddscasr <inputfile> <outputRoot> [-verbose]
      [-daemon] [-dailyFiles] [-semaphore=<filename>] [-save] [-restore] [-logFile=<filename>]
      [-runControlPV={string=<string>|parameter=<string>},pingTimeout=<value>,pingInterval=<value>]
      [-runControlDescription={string=<string>|parameter=<string>}] [-unique] [-outputFilePV=<pvname>]
      [-pidFile=<pidFile>] [-casavePV=<string>] [-interval=<seconds>] [-pipe=[input|output]]
      [-numerical] [-waveform=[rootname=<string>][,directory=<string>]]
      [-outputFilePV=<pvname>] [-casavePV=<string>]
  ```

- **files:**
  - **input file:**
    The input file is an SDDS file with a string column. For saving snapshot, the input file contains at least one string column - ControlName. For restoring snapshot, the input file contains at least two strin columns - ControlName and ValueString, where ValueString is the value of the PVs to be restored.
  - **output file:**
    The output file only exists for saving snapshot. The output file contains everything in the input file, except that the ValueString column (if the input file has ValueString column) is update. And three more columns are created if they do not exist in the input file and their values are updated.
    * ValueString: the values of the PVs given in the input file.
    * IndirectName: its value is - for scalar PV or PVname for waveform PV.
    * CAError: its value is y if error occurred during channel access for corresponding PV or n if no error occurrs.

- **switches:**
  - `inputFile` — inputFile name (SDDS file).
  - `outputRoot` — output file or root if -dailyFiles option is specified.
  - `pipe[=input][,output]` — The standard SDDS Toolkit pipe option. -dailyFile option is ignored if -pipe=out or -pipe option is present.
  - `semaphore` — specify the flag file for indicating CA connection completence. the current outputFile name is written to semaphore file.
  - `daemon` — run in daemon mode, that is, stay in background and start running whenever signal event arrived until terminating signal received.
  - `save` — read the values of PVs given in the input file and write to the output file.
  - `restore` — set the values of process variables given in the input file. -save and -restore can not be given in the same time.
  - `logFile` — file for logging the printout information.
  - `pidFile` — provide the file to write the PID into.
  - `interval` — the interval (in seconds) of checking monitor pv or the sleeping time if no signal handling arrived.
  - `pendIOTime` — specifies maximum time to wait for connections and return values. Default is 30.0s. It is important to give enough pendIOTime, otherwise, the CA connection could not set up and errors would occur. For SR request file, normally 100 seconds is approriate.
  - `casavePV` — a monitor pv to turn on/off sddscasr. Whenever the casavePV is set to 1, sddssave make a save or restore and change the pv back to 0.
  - `outputFilePV` — a string pv to store the output file name (exclude the path).
  - `unique` — remove all duplicates but the first occurrence of each PV from the input file to prevent channel access errors when setting pv values.
  - `description` — specify SnapshotDescription parameter of output file.
  - `runControlPV=string=<string>|parameter=<string>[,pingTimeout=<value>,pingInterval=<value>]` — specifies the runControl PV record.
  - `runControlDescription` — specifies a string parameter whose value is a runControl PV description record.
  - `numerical` —return a numerical string value rather than a string for enumerated types.
  - `waveform=rootname=<string>[,directory=<string>]` —provide the waveform rootname and directory in restore option. By default, the directory is pwd, the rootname is the SnapshotFilename parameter in the input file.
  - `outputFilePV` — a charachter-waveform pv to store the output file name with full path.
  - `casavePV` — a monitor pv to turn on/off sddscasr. When the pv changed from 0 to 1, sddssave make a save and change the pv back to 0.
- **see also:**
  - `burtrb` (2.1)
  - `burtwb` (2.2)
- **author: H. Shang, ANL**

2.8 sddscontrollaw

- **description:** `sddscontrollaw` performs simple feedback on process variables. The input file defines a gain matrix in a simple control law equation. The set of process variables for measurement are given by the names of the numerical data columns, and the set of process variables for control are given by a string column. By default, the feedback tries to regulate the values of the measurement to zero. The output file is a log of all process variables during the feedback. For robustness, a file of tests for a set of process variables may be defined so that the feedback may be suspended when any tests fail.

- **example:** The trajectory of the LTP beamline and the energy of the linac upstream of the LTP beamline is controlled with this command:

```
sddscontrollaw LTP.InvR12 LTPfeedback.out -interval=5 -steps=3600 \
        -gain=0.75 -warning
```

  where the contents of the file `LTP.InvR12` are

```
SDDS1
&column
 name = "CorrectorNames",  type = "string", &end
! LTP:PH* are readbacks of beam position monitors
&column
 name = "LTP:PH1",  type = "double", &end
&column
 name = "LTP:PH2",  type = "double", &end
&column
 name = "LTP:PH3",  type = "double", &end
&column
 name = "LTP:PH4",  type = "double", &end
&data
 mode = ascii, &end
! LTP:H[124] are dipole steering magnets.
LTP:H1          1.45e-01  7.95e-02  1.84e-01 -3.70e-02
LTP:H2          0.00e-00  2.12e-01  3.34e-01  8.39e-02
! Sled timing controls the linac energy.
L5:SledTiming   0.00e-00  0.00e-00  8.25e-03  9.45e-03
LTP:H4          0.00e-00  0.00e-00 -9.81e-03  1.44e-01
```

- **synopsis:**

```
usage: sddscontrollaw <inputfile> [<outputfile>]
       -controlQuantityDefinition=<file>
       [-gain=<real-value>] [-interval=<real-value>]
       [-steps=<integer=value>] [-updateInterval=<integer=value>]
       [{-integration | -proportional}] [-holdPresentValues]
       [-verbose] [-dryRun]
       [-testValues=<SDDSfile>] [-warning]
       [-ezcaTiming=<timeout>,<retries>] [-groupEzca]
Perform simple feedback on APS control system process variables using ezca calls.
```

- **files:**
  - **input file:**
    The input file is an SDDS file with a string column and at least one numerical column. The first string column encountered gives the list of control process variables, the other string columns being ignored. The names of the numerical columns are the measurement process variables. The numerical data columns form a matrix which will be used in a simple control-law equation such as
    $$\mathbf{u_n} = -K\mathbf{x_n} \tag{1}$$
    or
    $$\mathbf{u_n} = \mathbf{u_{n-1}} - K\mathbf{x_n}. \tag{2}$$
    The quantity x_n is the vector of measurement process variable values at step n, K is the matrix, u_n is the vector of control process variables calculated to reduce the values of the measurement process variables on the next iteration. The first equation refers to proportional control, while the second one refers to integral control. For trajectory or orbit correction in accelerators, where x_n are beam position monitor readbacks, and u_n are steering magnet setpoints, one chooses the integral control equation.
  - **control quantity definition file:**
    An optional control quantity definition file may be specified. This file allows one to use names in the matrix input file that are not really process variables, but more simplified and descriptive names. This situation can occur if the matrix file is obtained from post-processing of `sddsexperiment` output data, a common way to generate the correction matrix empirically. The control quantity definition file is a cross-reference file which `sddscontrollaw` uses to match the input file names to real PV names. Two columns are required:
    * `SymbolicName` — String. Data must match all column names in the input file, and string column data in the input file.
    * `ControlName` — String. Corresponding PV names.
  - **test values file:**
    To make `sddscontrollaw` more robust, one can implement tests on any process variable, not necessarily those involved in the feedback itself. If any of the tests fail, then the feedback is suspended until the test suceeds. The test consist of checking whether a process variable is within a specified range or not. The `testValues` file has three required columns and one optional one:
    * `ControlName` — Required string column. PV names to test.
    * `MinimumValue`, `MaximumValue` — Required double columns. Defines a valid range for corresponding PVs. An error is generated when `MinimumValue` > `MaximumValue`.
    * `SleepTime` — Optional double. Specifies sleep (or pause) time before attempting another test.
  - **output log file:**
    The output file contains one data column for each process variables defined in the input file. By default, the data type is float (single precision). One row is written at every time step.

    Two time columns and a step column are defined:

- * `Time` — Double. Elapsed time of readback since the start of epoch.
  * `ElapsedTime` — Double. Elapsed time of readback since the start of execution.
  * `TimeOfDay` — Double. System time in units of hours. The time does not wrap around at 24 hours.
  * `Step` — Long. Step number.

  There are two parameters defined:

  * `TimeStamp` — String. Time stamp for file.
  * `Gain` — Double. Gain factor specified on the commandline.

- **switches:**
  - `-controlQuantityDefinition=<file>` — an optional cross-reference file which matches the simplified names used in the inputfile to real PV names. Required string column are `SymbolicName` and `ControlName`. The data in column `SymbolicName` must match all column names in the input file, and string column data in the input file. Data in `ControlName` must be valid process variable names.
  - `-gain=<real-value>` — quantity multiplying the inputfile matrix. If the gain matrix is exactly equal to the inverse response matrix between the control and measurement process variables, then this value should be less than one. If instability occurs in the feedback, then the program should be re-run with a reduced value for `gain`.
  - `-interval=<real-value>` — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
  - `-steps=<integer-value>` — Number of iterations for the feedback before normal exiting.
  - `-updateInterval` — Obsolete.
  - `-integral | -proportional` — Switch the control law to either integral or proportional control; use integral control for orbit correction; the default control law is integral control.
  - `-holdPresentValues` — Causes regulation of the readback variables to the values at the start of the running. In this case, the control law equations are

    $$\mathbf{u_n} = -K(\mathbf{x_n} - \mathbf{x_0}) \tag{3}$$

    or

    $$\mathbf{u_n} = \mathbf{u_{n-1}} - K(\mathbf{x_n} - \mathbf{x_0}). \tag{4}$$

    where x_0 is the vector of initial values of measurement PVs (i.e. values before running the program).
  - `-verbose` — prints out a message when data is taken.
  - `-dryrun` — readback variables are read, but the control variables aren't changed.
  - `-testValues=<file>` — sdds format file containing minimum and maximum values of PV's specifying a range outside of which the feedback is temporarily suspended. Column names are ControlName, MinimumValue, MaximumValue, SleepTime (optional).
  - `-warning` — prints warning messages.

- `-ezcaTiming[=<timeout>,<retries>]` — Sets EZCA timeout and retry parameters.

- **see also:**
  - `sddsexperiment` (2.9)
- **author: L. Emery, H. Shang, R. Soliday, ANL**

2.9 sddsexperiment

- **description:** `sddsexperiment` varies process variables and measures process variables, with optional statistical analysis. An input file of namelist commands gives the specific instructions. The results are recorded in one or more SDDS files.

- **example:** The strength of a beamline horizontal corrector (LTP:H1) is varied while the readbacks at a horizontal beam position monitor (LTP:PH1) are recorded. The output file is LTP:H1.sdds.

```
sddsexperiment LTP:H1.exp LTP:H1.sdds
```

  where the contents of the file `LTP:H1.exp` are

```
&measurement    control_name = "LTP:PH1:x",
         column_name="LTP:PH1:x", units=mm,
         number_to_average = 10,
&end

&variable control_name = "LTP:H1:CurrentAO",
        column_name="LTP:H1:CurrentAO"
! the corrector is varied in 5 steps from -1.0 to 1.0 amps.
        index_number = 0, index_limit = 5,
        initial_value = -1.0, final_value = 1.0,
&end

&execute
        post_change_pause=4,
        intermeasurement_pause=1
&end
```

  where the line starting with a "!" is a comment.

- **synopsis:**

```
usage: sddsexperiment <inputfile> [<outputfile-rootname>]
[-echoinput] [-dryrun] [-summarize] [-verbose]
[-ezcaTiming=<timeout>,<retries>]
```

- **files:**
  - **input file:**
    The input file consists of namelist commands that set up and execute the experiment. The functions of the commands are described below.
    * `variable` — specifies a process variable to vary, and the range and steps of the variation. More than one variable command may be defined, so that many process variables may vary at a time. When an arbitrary sequence of setpoint values is required, the setpoints can be read in from a file.

* `measurement` — specifies a process variable to measure at each step during the experiment. Instead of using many `measurement` namelist, one can optionally use a sddsmonitor-compatible file.
* `execute` — start executing the experiment. One group of variable, measurement and execute commands may follow another in the same file for multiple experiments.
* `erase` — deletes previous variable or measurement setups.
* `list_control_quantities` — makes a cross-reference file for process variable names and column names of the data file.
* `system_call` — specifies a system call (usually a script) to be executed either before a measurement or before setting a process variable.

The following text describes all the namelist commands and their respective fields in more detail. The command definition listing is of the form

```
&<command-name>
    <variable-type> <variable-name> = <default-value>
    .
    .
    .
&end
```

where the part `<variable-type>`, which doesn't appear in an actual command, is used to illustrate the valid type of the value. The three valid types are:

* `double` — for a double-precision type, used for most physical quantity variables,
* `long` — for an integer type, used for indexes and flags.
* `STRING` — for a character string enclosed in double quotes.

An actual namelist in an input file should look like this:

```
&<command-name>
    [<variable-name> = <value>,]
    ...
&end
```

In the namelist definition listings the square brackets denotes an optional component. Not all variables need to be defined – the defaults may be sufficient. Those that do need to be defined are noted in the detailed explanations. The only variables that don't have default values in general are string variables.

variable

* function: Specifies a process variable to vary, and the range and steps of the variation. Values of variables at each measurement step are written to an SDDS output file. The readback-related fields are used to confirm that the physical device has responded to a setpoint command at every step (and substep) within some tolerance. Readback is enabled when `readback_attempts` and `readback_tolerance` are defined with non-zero positive values.
  When an arbitrary sequence of setpoint values is required (say a binary sequence), the values can be read in from an SDDS file specified by the `values_file` field. The fields of `&variable` associated for the range and steps are ignored in this case.
  With multiple `variable` commands, variables may be varied in a multi-dimensional grid. For example, variables may be varied independently of each other, or some groups of variables may vary together forming one axis of a multi-dimensional grid (see item `index_number`).

```
&variable
        STRING control_name = NULL
        STRING column_name = NULL
        STRING symbol = NULL
        STRING units = "unknown"
        double initial_value = 0
        double final_value = 0
        long relative_to_original = 0
        long index_number = 0
        long index_limit = 0
        STRING function = NULL
        STRING values_file = NULL
        STRING values_file_column = NULL
        long substeps = 1
        double substep_pause = 0
        double range_multiplier = 1
        STRING readback_name = NULL
        double readback_pause = 0.1
        double readback_tolerance = 0
        long readback_attempts = 10
        long reset_to_original = 1
&end
```

* `control_name` — Required. Process variable name to vary.
* `column_name` — Required. Column name for the variable data recorded in the output file.
* `symbol` — Optional. Symbol field for the above column definition of the variable data.
* `units` — Optional. Units field for the above column definition of the variable data.
* `initial_value` — Required. The initial value of the process variable in the variation.
* `final_value` — Required. The final value of the process variable in the variation.
* `index_number` — Required. The counter (or index) number with which the defined variation is associated. In a `sddsexperiment` run, counters must be defined in an

increasing order with no gaps starting from counter 0. That is, the first variable command of the file must have index_number = 0. The second variable command must have index_number = 0 or 1. In the former case, the two variables will move together with the same number of steps according their respective initial_value and final_value. In the latter case, the two variables will vary independently of each other with possibly different number of steps in a 2-dimensional grid.

Counter number n is nested within counter $n+1$. Therefore it might be efficient to assign devices with slower response times to higher index_number counter.

* index_limit — Normally required. Number of steps in the variation. Measurements are taken at each step. When more than one variable is associated with the same counter, only the index_limit of the first variable definition for that counter need to be defined. If index_limit is defined in variable commands of the same index_number value, then the first index_limit remain in force.
* relative_to_original — Optional. If non-zero, then the variation range is defined relative to the original process variable value (i.e. the value prior to running the program).
* range_multiplier — Optional. Factor by which the range, final_value - initial_value, is multiplied. New values of initial_value and final_value are calculated while keeping the midpoint of the range the same.
* function — Optional. A string of rpn operations used to transform the range specified by initial_value, final_value, and index_limit. For convenience, a few values are pushed onto the stack and are available to the user: the original value of the process variable, and the calculated grid value for the process variable on the current step or substep. The calculated values are recorded in the output file. The environment variable RPN_DEFNS is used to read a rpn definition file at the start of the execution of sddsexperiment.
* values_file — Optional. An SDDS data file containing setpoints for the variable. This is useful is one has arbitrary setpoints values to apply. The values of the fields initial_value, final_value, _substeps, range_multiplier and index_limit are ignored.

  One can have other variable namelists with the same index_number that don't use a file for the values. The default index_limit of the other variable will be set to the number of setpoint in the values file. Thus the values in the file and the values calculated for the other variable will vary together with the same number of steps.
* values_file_column — Required when values_file is specified. values_file_column gives the column name of the setpoints data in file values_file.
* substeps — Optional. If greater than one, the steps are subdivided into this number. Measurements are not made at substeps. Substeps are useful when the physical device associated with the process variable cannot reliably make steps as large as those that might be defined with initial_value, final_value, and index_limit.
* substep_pause — Optional. Number of seconds to pause after the variable change of each substeps.
* readback_name — Optional. Readback process variable name associated with control_name. The default value for readback_name is control_name.
* readback_tolerance — Optional. Maximum acceptable absolute value of the difference between the process variable setpoint and its readback. A positive value is required in order to enable readbacks.

* readback_pause — Optional. Number of seconds to pause after each reading of the readback_name process variable. This pause time is in addition to other pauses defined.
* readback_attempts — Optional. Number of allowed readings of the readback_name process variable and readback pauses after a variable change has occured. After this number of readings, the program exits. The first readback is attempted immediately (i.e. no pause) after sending a setpoint command to the control_name. A positive value is required in order to enable readbacks.
* reset_to_original — Optional. A value of 1 means that the variable is reset to its original value when the experiment terminates normally or abnormally.

measurement_file

* function: specifies a SDDS data file containing the names of the process variables to measure at each step during the experiment.

```
&measurement_file
        STRING filename = NULL
        long number_to_average = 1
        long include_standard_deviation = 0
        long include_sigma = 0
        double lower_limit = 0
        double upper_limit = 0
        long limits_all = 0
&end
```

* `filename` — Required. SDDS file containing measurement process variables. The required and optional column definitions closely resembles those used in `sddsmonitor`. The columns are:
  - `ControlName` — Required string data for the process variable names.
  - `ReadbackName` — Optional string data for the measurement data columns names of the experiment output file. If the column is not present, then the experiment output file's columns names uses the measurement process variable names themselves.
  - `NumberToAverage` — Optional long data giving the number of measurements to average at each step of the experiment. The average value of the measurements for each process variable is written to the output file. Therefore individual readings are not recorded.
  - `IncludeStDev` — Optional long data. If non-zero, then the standard deviations of measurements are calculated and a column in the output file for this quantity is generated.
  - `IncludeSigma` — Optional long data. If non-zero, then the sigma of measurements are calculated and a column in the output file for this quantity is generated.
  - `LowerLimit`, `UpperLimit` — Optional double data. Must have both or neither. Specifies a range outside of which the measurement data is to be rejected, and the measurement be retaken. If the number of invalid measurements (reset to 0 at each measurement step) equals or exceeds the value of `allowed_limit_errors` (default of 1) in command `execute`, then the program aborts. The average values written to the output file excludes measurements outside this range.
  - `LimitsAll` — Optional long data. If non-zero for a particular PV, then out-of-range data for this PV causes all other PVs to be remeasured. By default, only the out-of-range PV is remeasured.

  Only the first data page of file `filename` is read in. For those optional columns above that are not defined, then the following `measurement_file` fields will act as defaults (note the different spellings):`number_to_average`, `include_standard_deviation`, `include_sigma`, `lower_limit`, `upper_limit`, `limits_all`.

measurement

* function: specifies a process variable to measure at each step during the experiment. This command is compatible with `measurement_file`, as both commands merely adds to an internal list of measurement PV.

```
&measurement
        STRING control_name = NULL
        STRING column_name = NULL
        STRING symbol = NULL
        STRING units = "unknown"
        long number_to_average = 1
        long include_standard_deviation = 0
        long include_sigma = 0
        double lower_limit = 0
        double upper_limit = 0
        long limits_all = 0
&end
```

* `control_name` — Required. Process variable name to measure.
* `column_name` — Required. Column name for the measurement data recorded in the output file.
* `symbol` — Optional. Symbol field for the above column definition of the measurement data.
* `units` — Optional. Units field for the above column definition of the measurement data.
* `number_to_average` — Optional. Number of readings to average. The output data is the average value. The individual readings are not recorded.
* `include_standard_deviation` — Optional. Calculate the standard deviation (a measure of the distribution) of the measurement, and generate a column in the output file for this quantity.
* `include_sigma` — Optional. Calculate the sigma (error of the mean) of the measurement, and generate a column in the output file for this quantity.
* `lower_limit`, `upper_limit` — Optional. Defines a range of validity for the individual measurements. If the number of invalid measurements (reset to 0 at each measurement step) equals or exceeds the value of `allowed_limit_errors` (default of 1) in command `execute`, then the program aborts. The average values written to the output file excludes measurements outside this range.
* `limits_all` — Optional. If non-zero, then out-of-limits data for this PV causes all PVS to be remeasured. By default, only the out-of-limit PV is remeasured.

execute

* function: start executing the experiment. Some global parameters are defined here.

```
&execute
        STRING outputfile = NULL
        double post_change_pause = 0
        double intermeasurement_pause = 0
        double rollover_pause = 0
        long post_change_key_wait = 0
        long allowed_timeout_errors = 1
        long allowed_limit_errors = 1
        double outlimit_pause = 0.1
        long repeat_reading = 1
        double post_reading_pause = 0.1
        double ramp_pause = 0.25
        long ramp_steps = 10
&end
```

* `outputfile` — Optional. SDDS output file of variable and measurement data. The string may contain the string "%s" which is substituted with the rootname value of the command line. If rootname is not given, then the root of the intput file is substituted . If `outputfile` is null, then the root of the input file is used for the output file. See description of output file below.
* `post_change_pause` — Optional. Number of seconds to pause after each change before making measurements.
* `intermeasurement_pause` — Optional. Number of seconds to pause between each measurement. Individual measurements for averaging are taken at this interval.
* `rollover_pause` — Optional. Number of seconds to pause after a counter has reached its upper limit, and must rollover to zero. This allows any physical devices associated with the counter to settle after a change equal to the total range of the variation.
* `post_change_key_wait` — Optional. If non-zero, then wait for a key press after making variable changes but before taking measurements. A prompt is given.
* `allowed_timeout_errors` — Optional. Number of timeout ezca errors allowed before aborting the program.
* `allowed_limit_errors` — Optional. Number of invalid range measurement errors allowed before aborting the program. The valid range of a measurement is specified in the `measurement` command.
* `outlimit_pause` — Optional. Number of seconds to pause after an invalid range measurement error occured. This is to permit equipment time to recover from whatever glitch caused the out-of-limit reading.
* `repeat_reading` — Optional. The measurements and statistical analyses are repeated this number of times for each variable settings. A row of data is written to the output file for each repitition.
* `post_reading_pause` — Optional. Number of seconds to pause after taking a set of measurements and making a statistical analysis. If measurements are repeated then the pause is repeated after each set of measurements.

* `ramp_steps` — Optional. Number of steps in the variables PV ramp which occurs at the start and the end of the experiment.
  Ramping is necessary for some devices that do not respond well to large changes to their setpoints. Ramping is done at the start of the experiments to slowly change the variable PVs from their current values to their initial values. Another ramp is done at the end to slowly bring the variable PVs from their final values back the original values. Ramping back to original values is also done when the experiment aborts for some reason.
* `ramp_pause` — Optional. Time interval at each step of the variables PV ramp which occurs at the start and the end of the experiment. This is not the same variable as the pause between variable changes during the experiment.

erase

* function: deletes previous variable or measurement setups.

```
&erase
        long variable_definitions = 1
        long measurement_definitions = 1
&end
```

* `variable_definitions` — Optional. If non-null, then all the variable definitions are erased.
* `measurement_definitions` — Optional. If non-null, then all the measurement definitions are erased.

list_control_quantities

* function: makes a cross-reference file for process variable names and column names of the data file.

```
&list_control_quantities
         STRING filename = NULL
&end
```

* `filename` — Required. Name of file. Columns defined are `ControlName`, `SymbolicName`, and `ControlUnits`.

system_call

* function: specifies a system call (usually a script) to be executed repeatedly during the experiment.

```
&system_call
        STRING command = NULL
        long index_number = 0
        long index_limit = 0
        double post_command_pause = 0
        double pre_command_pause = 0
        long append_counter = 0
        STRING counter_format = "%ld"
        long call_before_setting = 0
        long call_before_measuring = 1
        STRING counter_column_name = NULL
&end
```

* `command` — Required. Name of shell command or script to execute.
* `index_number` — Required. Counter number with which the command will be associated. The command is executed when this counter is advanced or rolled over.
* `index_number` — Optional. Number of times the command is executed for the associated counter. This field is used only when the value of `index_number` above defines a new counter.
* `post_command_pause` — Optional. Number of seconds to pause after the completion of the command.
* `pre_command_pause` — Optional. Number of seconds to pause before executing the command.
* `append_counter` — Optional. If non-zero, the counter value is appended to the command when the system call is made.
* `counter_format` — Optional. Format for the counter if the counter value is appended to the command.
* `call_before_setting`, `call_before_measuring` — Optional. At a counter advance or rollover the command can be executed in one of three ways:
  - before both variable changes and measurements: `call_before_setting`=1, `call_before_measuring`=1
  - after variable changes and before measurements: `call_before_setting`=0, `call_before_measuring`=1
  - after both variable changes and measurements: `call_before_setting`=0, `call_before_measuring`=0

  If multiple measurements are required for averaging, the command is not executed between these measurements.
* `counter_column_name` — Optional. If non-null, a column in the output file with this name is defined. The values written to this column are the number of times the command had been called minus one. This value doesn't rollover with its associated counter.

exec_command

* function: executes one command one time. This is useful to the initial set-up of an experiment.

```
&exec_command
        STRING command = NULL
&end
```

* `command` — Command or script to run. The command is run immediately when the namelist command is processed

- **output file:**

  The output file contains one data column for each measurement and control process variable. The names of these data columns are taken from the `column_name` field of the respective `measurement` and `variable` commands. The data are written as double precision floating point numbers. In addition, some time columns and parameters are defined:

  * `Time` — Double column of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
  * `ElapsedTime` — Double column of elapsed time of readback since the start of the experiment.
  * `TimeOfDay` — Double column of system time in units of hours. The time does not wrap around at 24 hours.
  * `TimeStamp` — String parameter of time stamp for file.

- **control quantities list file:**

  This file contains process variable names, readback column names, and units as string data. This data can be used for cross-referencing. The columns defined in this file are:

  * `ControlName` — String column. Value of `control_name` field given in the `measurement` and `variable` commands.
  * `SymbolicName` — String column. Value of `column_name` field given in the `measurement` and `variable` commands.
  * `ControlUnits` — String column. Value of `units` field given in the `measurement` and `variable` commands.

- **switches:**

  - `-echoinput` — echos input file to stdout.
  - `-dryrun` — the “variable” process variables are left untouched during the execution. The “measurement” process variables are still read. The pauses are still in effect. System calls are not made.
  - `-summarize` — gives a summary of the experiment before executing it.
  - `-verbose[=very]` — prints out information during the execution such as notification of setting and reading process variables. The option `very` prints out the average measurement values.
  - `-ezcaTiming=<timeout>,<retries>` — sets EZCA timeout and retry parameters
  - `-describeinput` — Printouts the list of namelist commands and fields of the input file.

- **see also:**

  - `sddsvexperiment` (2.21)

- **author: M. Borland, L. Emery, H. Shang and R. Soliday ANL**

2.10 sddsfeedforward

- **description:** `sddsfeedforward` performs feedforward on process variables. Feedforward essentially means that one or more process variables are set according to a predetermined function of other process variables. In `sddsfeedforward`, the user supplies a series of tables that are used to interpolate values of a single actuator as a function of a single readback. Any number of tables may be given, so that any number of actuators may be driven by any number of readbacks. Each actuator is, however, driven by only one readback. One readback may drive any number of readbacks.

- **example:**

```
sddsfeedforward ID1GapCompensation.sdds -interval=10,s
```

where the file ID1GapCompensation.sdds contains the following data:

```
ReadbackName = ID1:Gap             ActuatorName = S1B:H1:CurrentAO
 ReadbackValue  ActuatorValue
------------------------------
       150.000          0.000
       100.000          0.000
        50.000          1.000
        40.000         10.000
        30.000         20.000
        20.000         35.000
        15.000         74.000
        11.000         99.000
```

At each iteration step, ID1:Gap is read and `sddsfeedforward` intepolates a new value for `S1B:H1:CurrentAO` based on this table, which it writes to the PV.

- **synopsis:**

```
sddsfeedforward <inputfile>
       [-interval=<real-value>] [-steps=<integer=value>]
       [-verbose] [-dryRun]
       [-averageOf=<number>[,interval=<seconds>]]
       [-testValues=<SDDSfile>]
       [-ezcaTiming=<timeout>,<retries>]
       [-runControlPV=string=<string>[,pingTimeout=<value>]] [-runControlDescription=<string>]
       [-CASecurityTest]
```

- **files:**
  - **input file:**
    The input file is an SDDS file with the following elements:
    * Parameters
      · `ReadbackName` — Required string parameter giving the name of the readback for this table. This is the process variable that is read from EPICS. Its value is used to interpolate a new value for the actuator.

      - `ActuatorName` — Required string parameter giving the name of the actuator for this table. This is the process variable that is set by the program according to interpolation with the readback values.
      - `ReadbackChangeThreshold` — Optional floating-point value giving the amount by which the process variable named in `ReadbackName` must change from the previous iteration's value in order for a change to be made for the actuator. Most useful if the readback is a little noisy and one doesn't want the actuator to be changed pointlessly. If the same process variable is named as readback for several actuators, then the smallest value of the change threshold is the one used.
      - `ActuatorChangeLimit` — Optional floating-point value giving the maximum amount by which the actuator should be changed in a single iteration. If several actuators have the same readback process variable, then they are limited together by the most constraining of the limits. In such a case, the vector of changes is scaled to bring all the changes within the respective limits.
    - Columns
      - `ReadbackValue` — Required floating-point column giving values of the readback for an interpolation table.
      - `ActuatorValue` — Required floating-point column giving values of the actuator for an interpolation table.
  - **test values file:**

    To make `sddsfeedforward` more robust, one can implement tests on any process variable, not necessarily those involved in the feedforward itself. If any of the tests fail, then the feedforward is suspended until the test suceeds. The test consist of checking whether a process variable is within a specified range or not. The `testValues` file has three required columns and one optional one:

    - `ControlName` — Required string column giving process variables names to test.
    - `MinimumValue`, `MaximumValue` — Required floating-point columns, defining a valid range for corresponding PVs. It is an error if `MinimumValue` $>$ `MaximumValue`.
    - `SleepTime` — Optional floating-point column, specifying sleep (or pause) time before attempting another test.

- **switches:**
  - `interval=<real-value>` — Specifies the interval in seconds between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
  - `steps=<integer-value>` — Number of iterations for the feedforward before normal exiting.
  - `verbose` — prints out a message when data is taken.
  - `dryRun` — readback variables are read and computations are performed, but the control variables aren't changed.
  - `averageOf=<number>[,interval=<seconds>]]` — Specifies averaging of the readback values, giving the number of readings to average and the interval in seconds between the readbacks.

- testValues=<file> — An SDDS file containing minimum and maximum values of PV's specifying a range outside of which the feedforward is temporarily suspended. The contents of the file are described above.
- ezcaTiming[=<timeout>,<retries>] — Sets EZCA timeout and retry parameters.
- runControlPV=string=<string>[,pingTimeout=<value>] — Gives the name of a process variable to use for workstation-independent process control via the Run Control facility and the value of run control ping time out.
- runControlDescription=<string> — Gives a text description to write to the Run Control record for process identification.
- CASecurityTest — Specifies that before writing values to EPICS, a test should be done of the state of Channel Access security on the process variables, to see if the process is permitted to write values.

- **see also:**
  - sddsexperiment (2.9)
- **author: M. Borland, ANL**

2.11 sddsgencontrolbits

- **description:** `sddsgencontrolbits` creates a file of BPM acquisition control bits from unsigned-long RAM data or from command line parameters. There is an option to send the RAM data to EPICS, introduced originally as a secondary feature. The separate bits of data of the output file can then be used in the `tcl/tk` GUI `MpBPMWaveformViewer` for generating waveforms displays of the control bits. It was later realized that this command tool with the EPICS waveform writing is really useful in script-based experiments where one or more aspect of the acquisition is modified in a systematic way.

  The BPM acquisition is controlled by an array of 3888 32-bit unsigned long integers loaded in as an EPICS waveform controls. Four BPMs can be controlled for up to 3888 consecutive samples in a repeated fashion. The 32 bits control the plane, accumulator and "use sample" switches for each BPM, and other global control switches. The control bits are shown in Figure 1 and explained in further detail in [3].

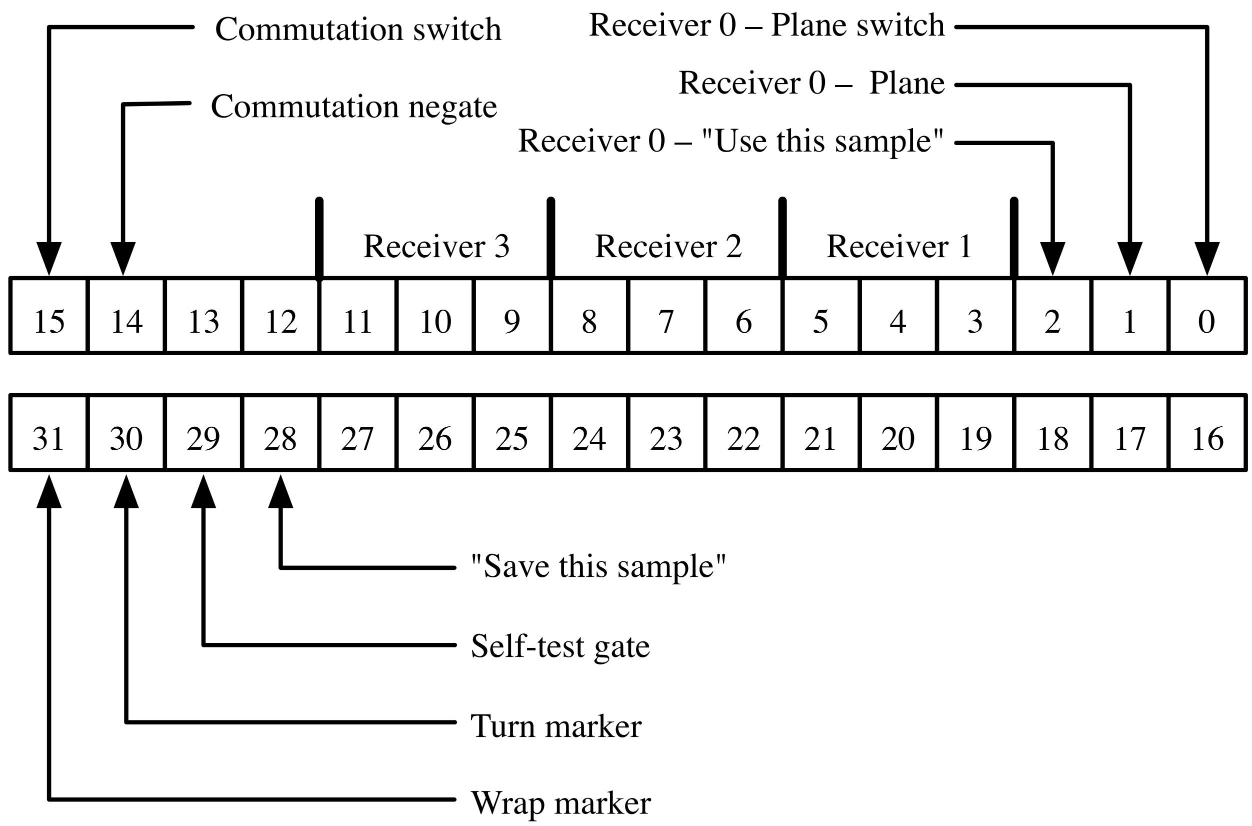

Figure 1: Acquisition control bit assignments. Courtesy of Eric Norum.

- **example:**

  The contents of the file `controlRAM` is converted to control bits file `RAMcontrolBits`.

  ```
  sddsgencontrolbits controlRAM RAMcontrolBits
  ```

  where the contents of the 3888-row file `controlRAM` are

3 columns of data:

NAME	UNITS	SYMBOL	FORMAT	TYPE	FIELD LENGTH	DESCRIPTION
Index	NULL	NULL	NULL	long	0	NULL
Waveform	NULL	NULL	NULL	ulong	0	NULL

and the contents of the file `RAMcontrolBits` are mostly short-integer columns

10 columns of data:

NAME	UNITS	SYMBOL	FORMAT	TYPE	FIELD LENGTH	DESCRIPTION
Index	NULL	NULL	NULL	long	0	NULL
PlaneSwitch	NULL	NULL	NULL	short	0	NULL
Accumulator	NULL	NULL	NULL	short	0	NULL
CommutationNegate	NULL	NULL	NULL	short	0	NULL
CommutationSwitch	NULL	NULL	NULL	short	0	NULL
Sample	NULL	NULL	NULL	short	0	NULL
TurnMarker	NULL	NULL	NULL	short	0	NULL
WrapMarker	NULL	NULL	NULL	short	0	NULL
SaveSample	NULL	NULL	NULL	short	0	NULL
SelfTest	NULL	NULL	NULL	short	0	NULL

This file `RAMcontrolBits` has two data pages, one page of 3888 rows for the RAM content array, and a second data page of 4096 rows to simulate what the FPGA transient recorder (i.e. "scope") would record. This data content is the same data as in the first page, but shifted by 2048 indices, and repeating in both directions to fill 4096 time slots. The second page seems to be superfluous, but it is needed to produce simulated waveforms in the GUI `MpBPMWaveformViewer`.

Presently the output only return data for one BPM at a time (`PlaneSwitch`,`Accumulator`, and `Sample`) by default the BPM in position 0 (from receiver positions 0,1,2,3). The column `Accumulator` is refered to in [3] as simply "Plane".

- **synopsis:**

```
sddsgencontrolbits <inputRAMFile> <outputFile>
          [-setRAMWaveformPV=<string>] [-controlRAMFile=<filename>] [-receiver=0|1|2|3]
sddsgencontrolbits {-RAMWaveformPV=<string> [-scopeArrayLength=<int>] |
          -scopeWaveformPV=<string> -turnsPerWrap=<int> [-RAMArrayLength=<string>] |
          -RAMWaveformPV=<string> -scopeWaveformPV=<string> } <outputFile>
          [-setRAMWaveformPV=<string>] [-controlRAMFile=<filename>]
sddsgencontrolbits  [-RAMArrayLength=<string>] [-scopeArrayLength=<int>] [-receiver=0|1|2|3]
          {-presetRAMconfiguration=<string>,file=<filename> |
           -planeMode=<string> -commutationMode=<string>]
           -sampleMode={single|continuous|bunchPattern=<string>}
           -bunchPatternFile=<filename> -samplesPerBunch=<int>
           -turnMarkerOffset=<int> -transitionDeadTime=<int> }
           [-P0Offset=<int>] [-receiver=0|1|2|3]
           [-setRAMWaveformPV=<string>] [-controlRAMFile=<filename>]
Creates a file of RAM control bits.
Optionally writes to a RAM acquisition control waveform.
```

```
Each usage represent one of three ways to supply input data:
1) input file of RAM data,
2) RAM or scope readback waveform PV, or
3) RAM configuration parameters
such as preset bunch pattern and others.
<inputRAMFile>      file containing the values of a RAM waveform record. The data
                    is an array of unsigned long integer.

-RAMWaveformPV      RAM waveform PV name.
-scopeWaveformPV    digital scope waveform PV name.
-RAMArrayLength     the length of control RAM waveform PV needed for generating control bits from
                    bunch pattern.
-scopeArrayLength   the length of digital scope waveform PV needed for generating control bits from
                    bunch pattern.
-turnsPerWrap       number of turns in each wrap;
                    "Simulated" scope readback control bits can normally be generated
                    from RAM long integer data. However for the inverse generation the quantity turnsPerWrap
                    is required. This option is required for the -scopeWaveformPV input method.

-presetRAMconfiguration  specifies a particular configuration from the file of RAM configuration presets.
                    A preset configuration consists of Label, PlaneMode, CommutationMode,
                    SampleMode, BunchPattern, SamplesPerBunch, TurnMarkerOffset, and TransitionDeadTime.
                    The preset string value must match one of the values of Label of the preset file.
                    The default file is /home/helios/oagData/sr/BPMcontrolRAM/presetPatterns.
                    If this option is provided, then the planeMode, commutationMode and sampleMode
                    options will be overridden.
-planeMode          the plane switch mode with values x, y, xy1, or xy2, which stands for
                    x plane, y plane, switch x/y per turn, or switch x/y every two turns.
                    (each turn has 324 time slots.)
-commutationMode    commutation switch mode with possible values of a, b, ab1, and ab2,
                    which stands for 0 degree, 180 degrees, switch between 0 and 180 every turn,
                    or switch  between 0 and 180 every two turns.
-sampleMode         sample mode with values continuous, single or pattern.
                    If "pattern" is selected, then the value of suboption bunchPattern must be given
                    and must match up with a bunch pattern defined in the preset bunch pattern file.
-bunchPatternFile   file that contains the properties of preset bunch patterns.
-transitionDeadTime dead time length in units of timing slots for which samples will not be taken.
-turnMarkerOffset   the offset of turn marker. The first turn marker starts from this offset, and
                    the others are spaced 324 time steps.
-samplesPerBunch    number of consectuive samples to be collected per bunch. Presently it
                    is the same for all bunches.
-scopeTriggerIndex  provides the offset between scope trigger and start of RAM array
                    (which is normally set at 2048 by PV S42B:scope:gtr:numberPTS).

The output options are the same in the three usages.
<outputFile>        file of the separated control bits of a RAM waveform record.
                    Columns are Index, PlaneSwitch, CommutationSwitch, Sample,
                    TurnMarker, P0Marker, WrapMarker, all short integers except for Index,
                    which is long integer.
                    The digital scope readback waveform record is simulated and is
                    included as the second data page.
-setRAMWaveformPV   if provided, the control RAM long integer data will be written to this PV.
                    Note that when used with -RAMWaveformPV input with the same PVs,
                    the same values are read and wrtten to the same PV.
```

```
-controlRAMFile     if provided, the control RAM long integer data will be written to this file.
-comment            provide comments for saving the control RAM into controlRAMFile .
-receiver           provide the receiver number to select from the control RAM when
                    creating the control bits file, which has room for only one set
                    of flags. When setting RAM all receivers have the same flags.
Program by H. Shang, ANL (EPICS 3.14.8.2, Mar 26 2007).
```

- **files:**
  - **input file:**
  - **output file:**
  - **bunch pattern file:**
  - **switches:**
    - `-setRAMWaveformPV` — need description.
- **see also:**
  - `sddsexperiment` (2.9)
- **author: L. Emery, H. Shang, R. Soliday, ANL**

2.12 sddsglitchlogger

- **description:** `sddsglitchlogger` reads values of process variables and writes them to a file at a specified time interval when a trigger occurrs. An input file defines the process variables to be monitored and/or the trigger parameters. An trigger file defines the the process variables that act as triggers.

- **example:** The position readbacks of PMs in linac are monitored with the command below.

```
sddsglitchlogger SRvac.mon . -time=24,hours -interval=1,minute
```

  "." specifies the directory of the outputfile is current directory,the file name is generated as related to the date. where the contents of the file `SRvac.mon` are

```
SDDS1
&parameter name=TriggerControlName type=string &end
&parameter name=MajorAlarm type=short &end
&parameter name=MinorAlarm, type=short &end
&parameter name=NoAlarm type=short &end
&parameter name=OutputRootname type=string &end
&column name=ControlName type=string  &end
&column name=ReadbackName type=string &end
&column name=ReadbackUnits type=string &end
&data mode=ascii no_row_counts=1 &end
!page 1
soliday:PM1:X:positionM
0
0
1
PM
soliday:PM1:X:positionM  soliday:PM1:X:positionM  mm
soliday:PM1:Y:positionM  soliday:PM1:Y:positionM  mm
soliday:PM2:X:positionM  soliday:PM2:X:positionM  mm
...
```

- **synopsis:**

```
usage: sddsglitchlogger <input> <outputDirectory>|<outputRootname> [-triggerFile=<filename>]
        [-sampleInterval=<secs>] [-time=<real-value>[,<time-units>]
        [-circularBuffer=[before=<number>,][after=<number>]]
        [-holdoffTime=<seconds>] [-autoHoldoff]
        [-inhibitPV=name=<name>[,pendIOTime=<seconds>][,waitTime=<seconds>]]
        [-conditions=<filename>,{allMustPass | oneMustPass}]
        [-verbose] [-watchInput] [-lockFile=<filename>[,verbose]]
Writes values of process variables or devices to a binary SDDS file.
```

- **files:**
  - **input file:**
    The input file is an SDDS file with a few data columns required:

- * ControlName or Device — Required string column for the names of the process variables or devices to be monitored. Both column names are equivalent.
  * ReadbackName — Optional string column for the names of the data columns in the output file. If absent, process variable or device name is used.
  * ReadbackUnits — Optional string column for the units fields of the data columns in the output file.If absent, units are null.

  If triggerFile that gives the trigger information is not given, the input file should contain following parameters:

  * OutputRootname — Required string parameter for the root name of output file. Different OutputRootmames go to different output files. If different pages have the same OutputRootname, the columns defined in new pages are ignored. Only the trigger defined in newer page is counted to the same output.
  * TriggerControlName — Required string parameter for the name of process variable that acts as trigger.
  * MajorAlarm — Optional short parameter for alarm-based trigger. if nonzero, then severity of MAJOR on TriggerControlName, results in a buffer dump.
  * MinorAlarm — Optional short parameter for alarm-based trigger. if nonzero, then severity of MINOR on TriggerControlName, results in a buffer dump.
  * NoAlarm — Optional short parameter for alarm-based trigger. if nonzero, then severity of NOALARM on TriggerControlName, results in a buffer dump.
  * TransitionDirection — Optional short parameter for transition-based trigger. required if TransitionThreshold parameter is defined.
    · -1 transition of TriggerControlName from above threshold to below threshold. results in buffer dump.
    · 0 ignore transition-based triggers for this PV.
    · 1 transition from below threshold to above threshold results in buffer dump.
  * TransitionThreshold — Optional double parameter for transition-based trigger. required if TransitionDirection parameter is defined. It defines the threshold of a transition trigger.
  * GlitchThreshold — Optional double parameter for glitch-based trigger.
    · 0 ignore glitch-based triggers for this PV.
    · >0 absolute glitch level.
    · <0 -1*(fractional glitch level).
  * GlitchBaselineSamples — Optional long parameter for glitch-based trigger. It defines number of samples to average to get the baseline value for glitch determination. A glitch occurs when newReading is different from the baseline by more than GlitchThreshold or (if GlitchThreshold¡0) by —GlitchThreshold*baseline—.
  * GlitchBaselineAutoReset — Optional short parameter for glitch-based trigger. Normally (if there is no glitch) the baseline is updated at each step using baseline -¿ (baseline*samples+newReading)/(samples+1). After a glitch, one may want to do something different.
    · 1 After a glitch, the baseline is reassigned to its current value.
    · 0 The pre-glitch baseline is retained.
- **trigger file:**

The trigger file is an SDDS file with following columns, the meaning of these columns are the same as the parameters defined in input file, which are replaced by a trigger file:

* `TriggerControlName` — Required string column for the names of the process variables that act as triggers.
* `MajorAlarm` — Optional short column.
* `MinorAlarm` — Optional short column.
* `NoAlarm` — Optional short column.
* `TransitionThreshold` — Optional double column. (required for transition trigger exists)
* `TransitionDirection` — Optional short column.(required for transition trigger exists)
* `GlitchThreshold` — Optional double column.(required for glitch trigger exists)
* `GlitchBaselineSamples` — Optional long column.
* `GlitchBaselineAutoReset` — Optional short column

– **conditions file:**

The conditions file is an optional input file specified on the command line which lists conditions that must be satisfied at each time step before the data can be logged.

The file is like the main input file, but has numerical columns `LowerLimit` and `UpperLimit`. The minimal column set is `ControlName`, which contain the PV names, and the two limits columns above. Depending on comand line options, when any or all PV readback from this file is outstide the range defined by the corresponding data from `LowerLimit` and `UpperLimit`, none of the data of the input file PVs are recorded. When this situation occurs for a long period of time, the size of the output file doesn't grow, and it may appear that the monitoring process has somehow stopped. It is possible to check the program activity with the `touch` sub-option which causes the monitoring program to touch the output file at every step.

– **output file:**

If trigger file is not given, the output file name is: outputDirectory/OutputRootname-string here, there may be many output files depends how many pages and how many different OutputRootnames the input file has. If trigger file is given, the outputDirectory given in command line is actually the OutputRootname, the output file name is now (only one output in this case): outputDirectory-string In above, both string is generated by MakeDailyGenerationFilename().

The output file contains one data column for each process variables named in the input file. Time columns and other miscellaneous columns are defined:

* `Time` — Double column of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
* `TimeOfDay` — Float column of system time in units of hours. The time does not wrap around at 24 hours.
* `DayOfMonth` — Float column of system time in units of days. The day does not wrap around at the month boundary.
* `Step` — Long column for step number.
* `CAerrors` — Long column for number of channel access errors at each reading step.

Many time-related parameters are defined in the output file:

  * `TimeStamp` — String parameter for time stamp for file.
  * `PageTimeStamp` — String parameter for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The `PageTimeStamp` value for the new page is the creation date of the new page. The `TimeStamp` value for the new page is the creation date of the very first page.
  * `StartTime` — Double parameter for start time from the `C` time call cast to type double.
  * `YearStartTime` — Double parameter for start time of present year from the `C` time call cast to type double.
  * `StartYear` — Short parameter for the year when the file was started.
  * `StartJulianDay` — Short parameter for the day when the file was started.
  * `StartMonth` — Short parameter for the month when the file was started.
  * `StartDayOfMonth` — Short parameter for the day of month when the file was started.
  * `StartHour` — Short parameter for the hour when the file was started.

- **switches:**
  - `-triggerFile` — specifies the name of trigger file.
  - `-lockFile` — specifies the name of lock file. When this option is given, sddsglitchlogger uses the named file to prevent running multiple versions of the program. If the named file exists and is locked, the program exits. If it does not exist or is not locked, it is created and locked.
  - `-sampleInterval=<real-value>[,<time-units>]` — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
  - `-time=<real-value>[,<time-units>]` — Total time for monitoring. Valid time units are seconds, minutes, hours, and days. The completion time may be longer, because the time interval in not garanteed.
  - `-enforceTimeLimit` — Enforces the time limit given with the -time option.
  - `-circularBuffer` — Set how many samples to keep before and after the triggering event.
  - `-holdoffTime` — Set the number of seconds to wait after a trigger or alarm before accepting new triggers or alarms.
  - `-autoHoldoff` — Sets holdoff time so that the after-trigger samples are guaranteed to be collected before a new trigger or alarm is accepted.
  - `-verbose` — Prints out a message when data is taken.
  - `-pendIOtime=<value>` — sets the maximum time to wait for return of each value.
  - `-inhibitPV` — Checks this PV prior to each sample. If the value is nonzero, then data collection is inhibited. None of the conditions-related or other PVs are polled.
  - `-watchInput` – If it is given, then the programs checks the input file to see if it is modifed. If the inputfile is modified, then read the input files again and start the logging.

- `-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]` — Names an SDDS file containing PVs to read and limits on each PV that must be satisfied for data to be taken and logged. The file is like the main input file, but has numerical columns LowerLimit and UpperLimit.

  One of `allMustPass` or `oneMustPass` must be specified. It would make sense to use `allMustPass` in most monitoring applications. If `touchOutput` is present, then the output file is touched, even if no data is written. This way, one can determine by the time stamp of the file whether the monitoring job is still alive when the conditions fail for a long period of time. If `retakeStep` is present, then the value of `Step` in the output file is not incremented until the conditions pass, and data is written to the output file.

- **see also:**
  - `sddsmonitor` (2.15)
- **author: Hairong Shang, ANL**

2.13 sddslogger

- **description:** `sddslogger` reads values of process variables and writes them to a file at a specified time interval. One or more input files defines the process variables to be monitored.
- **example:** The pressure readbacks of storage ring ion pumps and the stored current are monitored with the command below.

```
sddslogger SRvac.mon SRvac.sdds -time=24,hours -sampleInterval=1,minute
```

where the contents of the file `SRvac.mon` are

```
SDDS1
&description &end
&column
 name = ControlName,  type = string, &end
&column
 name = ControlType,  type = string, &end
&column
 name = ReadbackUnits,  type = string, &end
&column
 name = ReadbackName,  type = string, &end
&data
 mode = ascii, no_row_counts=1 &end
! page number 1
S35DCCT:currentCC pv mA S35DCCT
VM:01:3IP1.VAL pv Torr VM:01:3IP1
VM:01:2IP2.VAL pv Torr VM:01:2IP2
VM:01:2IP3.VAL pv Torr VM:01:2IP3
...
```

- **synopsis:**

```
usage: sddslogger <SDDSinputfile1> <SDDSoutputfile1> <SDDSinputfile2> <SDDSoutputfile2>...
    [-generations[=digits=<integer>][,delimiter=<string>][,rowlimit=<number>][,timelimit=<secs>] | -dailyFiles]
    [-sampleInterval=<real-value>[,<time-units>]
    [-logInterval=<integer-value>
    [-steps=<integer-value> | -time=<real-value>[,<time-units>]]
    [-enforceTimeLimit] [-offsetTimeOfDay]
    [-verbose] [-singleshot{=noprompt | stdout}]
    [-precision={single|double}]
    -onerror={usezero|skiprow|exit} [-pendIOtime=<value>]
    [-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]]
Writes values of process variables or devices to a binary SDDS file.
```

- **files:**
  - **input file(s):**
    The input files are SDDS files with a few data columns required:

* `ControlName` or `Device` — Required string column for the names of the process variables or devices to be monitored. Both column names are equivalent.
* `ReadbackUnits` — Required string column for the units fields of the data columns in the output file.
* `ReadbackName` — Optional string column for the names of the data columns in the output file. If absent, process variable or device name is used.
* `Message` — Optional string column for the device read message. If a row entry in column `ControlName` is a process variable, then the corresponding entry in `Message` should be a null string.
* `ScaleFactor` — Optional double column for a factor with which to multiply values of the readback in the output file.
* `Average` — Optional long column. If value is non-zero the process variable will have its average value logged. Otherwise it will log the most recent value. values of the readback in the output file.
* `DoublePrecision` — Optional long column. If value is non-zero the process variable will be logged as a double-precision number. Otherwise it will be logged as a single-precision number.

– **conditions file:**

The conditions file is an optional input file specified on the command line which lists conditions that must be satisfied at each time step before the data can be logged.

The file is like the main input file, but has numerical columns `LowerLimit` and `UpperLimit`. The minimal column set is `ControlName`, which contain the PV names, and the two limits columns above. Depending on comand line options, when any or all PV readback from this file is outstide the range defined by the corresponding data from `LowerLimit` and `UpperLimit`, none of the data of the PVs in the input files are recorded. When this situation occurs for a long period of time, the size of the output files doesn't change, and it may appear that the monitoring process has somehow stopped. It is possible to check the program activity with the `touch` sub-option which causes the logging program to touch the output file at every step.

– **output file(s):**

The output files contains one data column for each process variable named in the corresponding input file. By default, the data type is float (single precision). Time columns and other miscellaneous columns are defined:

* `Time` — Double column of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
* `TimeOfDay` — Float column of system time in units of hours. The time does not wrap around at 24 hours.
* `DayOfMonth` — Float column of system time in units of days. The day does not wrap around at the month boundary.
* `Step` — Long column for step number.
* `CAerrors` — Long column for number of channel access errors at each reading step.

Many time-related parameters are defined in the output file:

* `TimeStamp` — String parameter for time stamp for file.

    * `PageTimeStamp` — String parameter for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The `PageTimeStamp` value for the new page is the creation date of the new page. The `TimeStamp` value for the new page is the creation date of the very first page.
    * `StartTime` — Double parameter for start time from the `C` time call cast to type double.
    * `YearStartTime` — Double parameter for start time of present year from the `C` time call cast to type double.
    * `StartYear` — Short parameter for the year when the file was started.
    * `StartJulianDay` — Short parameter for the day when the file was started.
    * `StartMonth` — Short parameter for the month when the file was started.
    * `StartDayOfMonth` — Short parameter for the day of month when the file was started.
    * `StartHour` — Short parameter for the hour when the file was started.

- **switches:**
  - `-generations[=digits=<integer>][,delimiter=<string>]` — The output is sent to the file `<SDDSoutputfile>-<N>`, where `<N>` is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating `<N>`, so that the first generation number is 0001.
  - `-sampleInterval=<real-value>[,<time-units>]` — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
  - `-logInterval=<interval>` — Specifies the number of sampling intervals to average before writing to the output file.
  - `-steps=<integer-value>` — Number of readbacks for each process variable before normal exiting.
  - `-time=<real-value>[,<time-units>]` — Total time for monitoring. Valid time units are seconds, minutes, hours, and days. The program calculates the number of steps by dividing this time by the interval. The completion time may be longer, because the time interval in not garanteed.
  - `-enforceTimeLimit` — Enforces the time limit given even if the expected number of samples has not been taken.
  - `-offsetTimeOfDay` — Adjusts the starting TimeOfDay value so that it corresponds to the day for which the bulk of the data is taken. Hence, a 26 hour job started at 11pm would have an initial time of day of -1 hour and a final time of day of 25 hours.
  - `-verbose` — Prints out a message when data is taken.
  - `-singleShot[=noprompt]` — a single read is prompted at the terminal and initiated by a `<cr>` key press. The time interval is disabled. With `noprompt` present, no prompt is written to the terminal, but a `<cr>` is still expected. Typing "q" or "Q" terminates the monitoring.
  - `-onerror={usezero|skiprow|exit}` — Selects action taken when a channel access error occurs. The default is using zero (`usezero`) for the value of the process variable with the channel access error, and resuming execution. The second option (`skiprow`) is to simply

throw away all the data for that read step, and resume execution. the third option is to exit the program.

  - `-pendIOtime=<value>` — Sets the maximum time to wait for connection to each PV.
  - `-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]` — Names an SDDS file containing PVs to read and limits on each PV that must be satisfied for data to be taken and logged. The file is like the main input file, but has numerical columns LowerLimit and UpperLimit.

    One of `allMustPass` or `oneMustPass` must be specified. It would make sense to use `allMustPass` in most monitoring applications. If `touchOutput` is present, then the output file is touched, even if no data is written. This way, one can determine by the time stamp of the file whether the monitoring job is still alive when the conditions fail for a long period of time. If `retakeStep` is present, then the value of `Step` in the output file is not incremented until the conditions pass, and data is written to the output file.

- **see also:**
  - `sddsmonitor` (2.15)
  - `sddsvmonitor` (2.22)
  - `sddswmonitor` (2.23)
  - `sddssnapshot` (2.18)
- **author: R. Soliday, M. Borland, H. Shang ANL**

2.14 sddslogonchange

- **description:** `sddslogonchange` records only the values of process variables that change. This reduces the output file size if process variables do not change often.

- **example:** Process variable setpoints from the storage ring are monitored with the command below.

```
sddslogonchange SR.loc SR.sdds -loginitial -watchInput -connectTimeout=120
```

  where the contents of the file `SR.loc` are

```
SDDS1
&column name=ControlName, type=string,  &end
&column name=Tolerance, format_string=%g, type=double,  &end
&column name=Description, type=string,  &end
&data mode = ascii, no_row_counts=1 &end
! page number 1
S1A:H1:CurrentAI.AOFF 0.0 ""
S1A:H2:CurrentAI.AOFF 0.0 ""
S1A:H3:CurrentAI.AOFF 0.0 ""
...
```

- **synopsis:**

```
usage: sddslogonchange <input> <output>
  [-timeDuration=<realValue>[,<time-units>]]
  [-append[=recover]]
  [-eraseFile]
  [-generations[=digits=<integer>][,delimiter=<string>]]
  [-dailyFiles]
  [-pendEventTime=<seconds>]
  [-durations]
  [-connectTimeout=<seconds>]
  [-explicit[=only]]
  [-verbose]
  [-comment=<parameterName>,<text>]
  [-requireChange[=severity][,status][,both]]
  [-inhibitPV=name=<name>[,pendIOTime=<seconds>][,waitTime=<seconds>]]
  [-includeTimeOfDay]
  [-offsetTimeOfDay]
  [-logInitialValues]
  [-logAlarms]
  [-watchInput]

Logs data for the process variables named in the ControlName
column of <input> to an SDDS file <output>.
```

- **files:**

- **input file:**

  The input file is an SDDS file with only one data column required:

  * `ControlName` — Required string column for the names of the process variables or devices to be monitored.
  * `ReadbackName` — Optional string column.
  * `ReadbackUnits` — Optional string column.
  * `Description` — Optional string column.
  * `RelatedControlName` — Optional string column.
  * `Tolerance` — Optional numeric column which contains a tolerance value to avoid logging small changes.

- **output file:**

  The output file contains parameters, arrays, and columns.

  * **Parameters**
  * `InputFile` — sddslogonchange input file.
  * `TimeStamp` — Time stamp for file.
  * `PageTimeStamp` — Time stamp for page.
  * `StartTime` — Start time.
  * `YearStartTime`
  * `StartYear`
  * `StartJulianDay`
  * `StartMonth`
  * `StartDayOfMonth`
  * `StartHour`

  * **Arrays**
  * `ControlName` — Control names of process variables
  * `ReadbackName` — Optional readback names of process variables
  * `ReadbackUnits` — Optional units of process variables
  * `AlarmStatusString` — Optional
  * `AlarmSeverityString` — Optional
  * `DescriptionString` — Optional
  * `RelatedControlName` — Optional

  * **Columns**
  * `ControlNameIndex` — Index of the ControlName array.
  * `Value` — Value of the process variable.
  * `Time` — Time stamp of the change in value.
  * `PreviousRow` — Previous row with an identical ControlNameIndex value.
  * `AlarmStatusIndex` — Optional
  * `AlarmSeverityIndex` — Optional

- * `Duration` — Optional
  * `RelatedValueString` — Optional
  * `ControlName` — Optional
  * `AlarmStatus` — Optional
  * `AlarmSeverity` — Optional
  * `Description` — Optional
  * `RelatedControlName` — Optional

- **switches:**
  - `-timeDuration=<realValue>[,<time-units>]` — Specifies time duration for logging. The default time units are seconds; you may also specify days, hours, or minutes.
  - `-offsetTimeOfDay` — Adjusts the starting TimeOfDay value so that it corresponds to the day for which the bulk of the data is taken. Hence, a 26 hour job started at 11pm would have initial time of day of -1 hour and final time of day of 25 hours.
  - `-append[=recover]` — Specifies appending to the file ¡output¿ if it exists already. If the recover qualifier is given, recovery of a corrupted file is attempted using sddsconvert, at the risk of loss of some of the data already in the file.
  - `-eraseFile` — Specifies erasing the file ¡output¿ if it exists already.
  - `-generations[=digits=<integer>][,delimiter=<string>]` — Specifies use of file generations. The output is sent to the file ¡output¿-¡N¿, where ¡N¿ is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating ¡N¿, so that the first generation number is 0001.
  - `-dailyFiles` — The output is sent to the file ¡output¿-YYYY-JJJ-MMDD.¡N¿, where YYYY=year, JJJ=Julian day, and MMDD=month+day. A new file is started after midnight.
  - `-pendEventTime=<seconds>` — Specifies the CA pend event time, in seconds. The default is 10.
  - `-durations` — Specifies including state duration and previous row reference data in output.
  - `-connectTimeout=<seconds>` — Specifies maximum time in seconds to wait for a connection before issuing an error message. 60 is the default.
  - `-explicit[=only]` — Specifies that explicit columns with control name, alarm status, and alarm severity strings be output in addition to the integer codes. If the "only" qualifier is given, the integer codes are omitted from the output.
  - `-verbose` — Specifies printing of possibly useful data to the standard output.
  - `-comment=<parameterName>,<text>` — Gives the parameter name for a comment to be placed in the SDDS output file, along with the text to be placed in the file.
  - `-requireChange[=severity][,status][,both]` — Specifies that either severity, status, or both must change before an event is logged. The default behavior is to log an event whenever a callback occurs, which means either severity or status has changed.
  - `-inhibitPV=name=<name>[,pendIOTime=<seconds>][,waitTime=<seconds>]` — Checks this PV periodically. If nonzero, then data collection is aborted.

- **see also:**

  - `sddslogger` (2.13)
  - `sddssnapshot` (2.18)

- **author: R. Soliday, ANL**

2.15 sddsmonitor

- **description:** `sddsmonitor` reads values of process variables and writes them to a file at a specified time interval. An input file defines the process variables to be monitored.
- **example:** The pressure readbacks of storage ring ion pumps and the stored current are monitored with the command below.

  ```
  sddsmonitor SRvac.mon SRvac.sdds -time=24,hours -interval=1,minute
  ```

  where the contents of the file `SRvac.mon` are

  ```
  SDDS1
  &description &end
  &column
   name = ControlName,  type = string, &end
  &column
   name = ControlType,  type = string, &end
  &column
   name = ReadbackUnits,  type = string, &end
  &column
   name = ReadbackName,  type = string, &end
  &data
   mode = ascii, no_row_counts=1 &end
  ! page number 1
  S35DCCT:currentCC pv mA S35DCCT
  VM:01:3IP1.VAL pv Torr VM:01:3IP1
  VM:01:2IP2.VAL pv Torr VM:01:2IP2
  VM:01:2IP3.VAL pv Torr VM:01:2IP3
  ...
  ```

- **synopsis:**

  ```
  usage: sddsmonitor <SDDSinputfile> <SDDSoutputfile>
      [-erase | -append[=recover] | -generations[=digits=<integer>][,delimiter=<string>]]
      [-steps=<integer-value> | -time=<real-value>[,<time-units>]]
      [-interval=<real-value>[,<time-units>]] [-updateinterval=<integer-value>]
      [-verbose] [-singleShot[=noprompt] [-precision={single|double}]
      [-oncaerror={usezero|skiprow|exit}] [-pendIOtime=<value>]
      [-ezcaTiming[=<timeout>,<retries>]]
      [-glitch=<controlname>[,message=<string>]{,delta=<value>|,fraction=<value>}\
  [,before=<number>][,after=<number>][,baseline=<number>][,holdoff=<seconds>]]
      [-trigger=<controlName>,level=<value>[,message=<string>][,slope={+ | -}]\
  [,before=<number>][,after=<number>][,holdoff=<seconds>][,autoArm]]
      [-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]]
      [-noezca] [-comment=<parameterName>,<text>]
      [-getUnits={force | ifBlank | ifNoneGiven}]\n\
  Writes values of process variables or devices to a binary SDDS file.
  ```

- **files:**
  - **input file:**
    The input file is an SDDS file with a few data columns required:
    * `ControlName` or `Device` — Required string column for the names of the process variables or devices to be monitored. Both column names are equivalent.
    * `Message` — Optional string column for the device read message. If a row entry in column `ControlName` is a process variable, then the corresponding entry in `Message` should be a null string.
    * `ReadbackName` — Optional string column for the names of the data columns in the output file. If absent, process variable or device name is used.
    * `ReadbackUnits` — Optional string column for the units fields of the data columns in the output file.If absent, units are null.
    * `ScaleFactor` — Optional double column for a factor with which to multiply values of the readback in the output file.
  - **conditions file:**
    The conditions file is an optional input file specified on the command line which lists conditions that must be satisfied at each time step before the data can be logged.

    The file is like the main input file, but has numerical columns `LowerLimit` and `UpperLimit`. The minimal column set is `ControlName`, which contain the PV names, and the two limits columns above. Depending on comand line options, when any or all PV readback from this file is outstide the range defined by the corresponding data from `LowerLimit` and `UpperLimit`, none of the data of the input file PVs are recorded. When this situation occurs for a long period of time, the size of the output file doesn't grow, and it may appear that the monitoring process has somehow stopped. It is possible to check the program activity with the `touch` sub-option which causes the monitoring program to touch the output file at every step.
  - **output file:**
    The output file contains one data column for each process variables named in the input file. By default, the data type is float (single precision). Time columns and other miscellaneous columns are defined:
    * `Time` — Double column of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
    * `TimeOfDay` — Float column of system time in units of hours. The time does not wrap around at 24 hours.
    * `DayOfMonth` — Float column of system time in units of days. The day does not wrap around at the month boundary.
    * `Step` — Long column for step number.
    * `CAerrors` — Long column for number of channel access errors at each reading step.

    Many time-related parameters are defined in the output file:
    * `TimeStamp` — String parameter for time stamp for file.
    * `PageTimeStamp` — String parameter for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The `PageTimeStamp` value for the new page is the creation date of the new page. The `TimeStamp` value for the new page is the creation date of the very first page.

* StartTime — Double parameter for start time from the C time call cast to type double.
* YearStartTime — Double parameter for start time of present year from the C time call cast to type double.
* StartYear — Short parameter for the year when the file was started.
* StartJulianDay — Short parameter for the day when the file was started.
* StartMonth — Short parameter for the month when the file was started.
* StartDayOfMonth — Short parameter for the day of month when the file was started.
* StartHour — Short parameter for the hour when the file was started.

- **switches:**
  - `-erase` — If the output file already exists, then it will be overwritten by `sddsmonitor`.
  - `-append[=recover]` — If the output file already exists, then append the new readings. The output file must have previously been generated by `sddsmonitor` using the same information in the input files. The `recover` option allows an attempt to recover the data using `sddsconvert` if the input file is somehow corrupted.
  - `-generations[=digits=<integer>][,delimiter=<string>]` — The output is sent to the file `<SDDSoutputfile>-<N>`, where `<N>` is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating `<N>`, so that the first generation number is 0001.
  - `-interval=<real-value>[,<time-units>]` — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
  - `-steps=<integer-value>` — Number of readbacks for each process variable before normal exiting.
  - `-time=<real-value>[,<time-units>]` — Total time for monitoring. Valid time units are seconds, minutes, hours, and days. The program calculates the number of steps by dividing this time by the interval. The completion time may be longer, because the time interval in not garanteed.
  - `-updateinterval` — Obsolete.
  - `-verbose` — Prints out a message when data is taken.
  - `-singleShot[=noprompt]` — a single read is prompted at the terminal and initiated by a `<cr>` key press. The time interval is disabled. With `noprompt` present, no prompt is written to the terminal, but a `<cr>` is still expected. Typing "q" or "Q" terminates the monitoring.
  - `-oncaerror={usezero|skiprow|exit}` — Selects action taken when a channel access error occurs. The default is using zero (`usezero`) for the value of the process variable with the channel access error, and resuming execution. The second option (`skiprow`) is to simply throw away all the data for that read step, and resume execution. the third option is to exit the program.
  - `-pendIOtime=<value>` — Obsolete.
  - `-ezcaTiming[=<timeout>,<retries>]` — Sets EZCA timeout and retry parameters.

- `-noezca` — Obsolete.
- `-glitch=<controlname>[,message=<string>],delta=<value>|,fraction=<value> [,before=<number>][,after=<number>][,baseline=<number>][,holdoff=<seconds>]` — Writes a buffer of PV readback values whenever the glitch PV (`<controlname>`) or device changes by some value. If `<controlname>` is a device, then the `message` field should be specified. A glitch is triggered if the control variable changes by the values of the `delta` or a `fraction` field with respect to an exponential average from `baseline` number of readings. The `before` and `after` fields give the number of readings recorded in a page before and after the glitch is triggered. Some buffers may be joined in one large page if the triggering events occur close together. Option `-oncaerror` is ignored.
- `-trigger=<controlName>,level=<value>[,message=<string>][,slope=+ | -] [,before=<number>][,after=<number>][,holdoff=<seconds>][,autoArm]` — Similar to glitch, except buffered data is recorded when the named PV exceeds the given `level` with the given `slope`. This is analogous to an oscilloscope trigger.
- `-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]` — Names an SDDS file containing PVs to read and limits on each PV that must be satisfied for data to be taken and logged. The file is like the main input file, but has numerical columns LowerLimit and UpperLimit.

  One of `allMustPass` or `oneMustPass` must be specified. It would make sense to use `allMustPass` in most monitoring applications. If `touchOutput` is present, then the output file is touched, even if no data is written. This way, one can determine by the time stamp of the file whether the monitoring job is still alive when the conditions fail for a long period of time. If `retakeStep` is present, then the value of `Step` in the output file is not incremented until the conditions pass, and data is written to the output file.
- `-comment=<parameterName>,<text>` — Gives the parameter name for a comment to be placed in the SDDS output file, along with the text to be placed in the file.
- `[-getUnits={force | ifBlank | ifNoneGiven}]` — Gets the units of quantities from EPICS. 'force' means ignore the ReadbackUnits data in the input, if any. 'ifBlank' means attempt to get units for any quantity that has a blank string given for units. 'ifNoneGiven' (default) means get units for all quantities, but only if no ReadbackUnits column is given in the file.

- **see also:**
  - `sddsvmonitor` (2.22)
  - `sddswmonitor` (2.23)
  - `sddssnapshot` (2.18)
- **author: L. Emery and M. Borland, ANL**

2.16 sddsoptimize

- **description:** `sddsoptimize` optimizes the RMS of a set of readback process variables by automatically varying setpoint process variables (or knobs composed of setpoint PVs), which have a physical influence on the readback process variables, through simplex or 1dscan method.

- **example:** The trajectory of the booster BPMs is controlled with this command:

```
sddsoptimize -measFile= booster.h.moni -varFile=vv -simplex=evaluations=50,divisions=12 \
             -knobFiles=booster.cokn -verbose
```

  where the contents of the file `booster.h.moni` are

```
SDDS1
&column
 name = "ControlName",  type = "string", &end
&column
 name = "ReadbackName",  type = "string", &end
&column
 name = "ReadbackUnits",  type = "string", &end
&data
 mode = ascii, no_row_counts=1 &end
oag:B1C0P1:ms:x B1C0P1:ms:x mm
oag:B1C0P2:ms:x B1C0P2:ms:x mm
oag:B1C1P1:ms:x B1C1P1:ms:x mm
oag:B1C1P2:ms:x B1C1P2:ms:x mm
.......
```

  the contents of the file `vv` are

```
SDDS1
&parameter name=PauseAfterChange, type=double, &end
&column name=ControlName, type=string,  &end
&column name=LowerLimit, type=double,  &end
&column name=UpperLimit, type=double,  &end
&column name=InitialChange, type=double,  &end
&data mode=ascii,no_row_counts=1,&end
oag:B1C1H:KickAO -2  2  2
oag:B1C2H:KickAO -2.500000000000000e+01  2.500000000000000e+01  2.000000000000000e-01
oag:B1C3H:KickAO -2.500000000000000e+01  2.500000000000000e+01  2.000000000000000e-01
B:h11cos         -25                     25                     0.1
B:h12sin         -25                     25                     0.1
B:h12cos         -25                     25                     0.1
.....
```

  the contents of the file `booster.cokn` are

```
SDDS1
&parameter name=ControlName, type=string, &end
&parameter name=KnobDescription, type=string, &end
&parameter name=Gain, type=double, &end
&parameter name=ControlType, type=string, &end
&parameter name=ControlUnits, type=string, &end
&parameter name=Filename, description="Name of file from which this page came", type=string, &end
&parameter name=NumberCombined, description="Number of files combined to make this file", type=long, &end
&column name=ControlName, type=string,  &end
&column name=Weight, type=double,  &end
&data mode=asc_ii,no_row_counts=1. &end
....
....
!.page 2
B:h11cos
h plane 11th harm. cos (1 mrad/click)
1.0
pv
rad
11HarmCosineh.cokn
8
oag:B1C0H:KickAO 0.99997256549446789
oag:B1C1H:KickAO -0.21202791415796657
oag:B1C2H:KickAO -0.93188756664225914
.....
.....
```

- **synopsis:**

```
usage: sddsoptimize -measFile=<filename> -measScript=<script>
        [-varScript=<scriptname>]
       -varFile=<filename> -knobFiles=<filename1> , <filename2>,...
       [-simplex=[restarts=<nRestarts>][,cycles=<nCycles>,]
       [evaluations=<nEvals>,][no1dscans][,divisions=<int>]]
       [-logFile=<filename>] [-verbose] [-tolerance=<value>] [-maximize]
       [-1dscan=[divisions=<value>,][cycles=<number>][,evaluations=<value>][,refresh]]
       [-target=<value>] [-testValues=file=<filename>[,limit=<count>]]
       [-runControlPV={string=<string>|parameter=<string>},pingTimeout=<value>,
         pingInterval=<value>
Perform optimization on APS control system process variables using simplex or 1dscan method.
```

- **files:**
  - **variable input file:**
    The variable input file is an SDDS file with one string column: ControlName, which is required and gives the list of control correctors (process variables or knobs). It also contains three double columns: LowerLimit, UpperLimit, IntialChange and InitialValue. InitialValue column is optional. Others are required. InitialChange column

specifies the initial changes to the correctors. Variable input file has one parameter –PauseBetweenReadings(double), which sets the waiting time in seconds between two settings of the correctors.

- **measurement file:**

  This file specifies the measurement to be optimized. It has four columns:

  * `ControlName` — Required string column. Gives the list of process variables to be controlled.
  * `ReadbackName` — string, optional.
  * `ReadbackUnits` — string, optional.
  * `Weight` – double, optional. Defines the weight of each PV contributed to RMS.

  It has three parameters:

  * `Tolerance` — double, sets the converging limit.
  * `NumberToAverage` — long, sets number of average for measurement PVs.
  * `PauseBetweenReadings` — double, sets interval between two readings.

- **knob file:**

  To make `sddsoptimize` more robust, one can implement optimizaion on knobs, that are composed of set point process varibles. The process variables that a knob contains are given in knob file, which contain following parameters and columns:

  * `ControlName` — Required string paramter. The name of knob, which acts as a corrector as other PVs do.
  * `ControlName` — Required string column. The names of PVs the knob specified above contains.
  * `Wieght` — Required double columns. Defines the weights of PVs that compose the knob.
  * `Gain` —Optional double paramter. The value of each PV the knob contains is value(PV)=value(knob)*Gain*Weigth(PV). When it is not given, set it to 1.
  * `ControlType` — Optional string paramter. Specifies the control type.
  * `ControlUnits` — Optional string parameter. Specifies the units of control PVs.
  * `KnobDescription` – Optional string parameter.
  * `Filename`— Optional string parameter. The name of the file where this knob comes from.
  * `Numbercombined` — Optional long parameter. The number of files combined. The resulted knob file contains all the information of the files combined. The content of each page is from the combined file specified in filename parameter.

- **output log file:**

  The output file contains one data column for each process variables defined in the variable input file. By default, the data type is double. One row is written at every evaluation. Also two more columns and two parameters are defined:

  * `EvalIndex` — Long Column. The index of evaluations.
  * `currentValue` — Double column. The RMS value of measurement at each evaluaion.
  * `variableFile` — String parameter. The name of input variable file.
  * `measurementFile` — String parameter. The name of input measurement file.

- **switches:**
  - `-varFile` — required, see above. Data in `ControlName`may be valid process variable names or knobs defined in knob files.
  - `-measFile` —specifies the name of measurement file. See above about its content.
  - `-knobFiles` —specifies list of knob files, which contain the knob correctors given in varFile. See above about its content.
  - `-measScript=<measScript>` — user given script for measuring PVs. Either -measScript or -measFile is given for measurement.
    tt¡measScript¿ is an executable script which is called by optimizer and outputs a value to the stand outputs. This value is the evaluation value and read by the optimizer.
  - `-varScript=<varScript>` — user given script for setting setpoint PVs or tags. If both -measScript and -varScript are given, there are no obvious ioc calls. The calling syntax of `<varScript>` is: `<varScript> -tagList <tagList> -valueList <valueList>` where `<varScript>` is an executable command, `<tagList>` is a list of setpoints or tag names supplied by the varFile and `<valueList>` is a list of values for setting the setpoints or tags. These values are calculated by optimizer in each evaluation.
  - `-simplex` — Give parameters of the simplex optimization. Each start or restart allows ¡nCycles¿ cycles with up to ¡nEvals¿ evaluations of the function. Defaults are 1 restarts, 1 cycles, and 100 evaluations (iterations). If no1dscan is given, then turn off the 1D scan function in simplex. Divisions gives the parameter of maximum divisions used in simplex. If repeat is given, then read the previous better point again to keep track with the noise. If there is not much noise, it should be turned off to have a better performance.
  - `-1dscan` — Give parameters of one dimensional scan optimization. Cycles gives the max. number of cycles (i.e. loops) of 1dscan and with an maximum number of evaluations. Divisons is the max. number of division applied in 1dscan. Either 1dscan or simplex method is used for optimize. repeat: read the measurement again if the variable PVs are set back to their previous values to keep track with noise.
  - `-verbose` — Specifies printing of possibly useful data to the standard output.
  - `-target` — the target value for RMS of measurement PV (or PVs).
  - `-tolerance` — tolerance should be given if measScript is used, otherwise, a default value of 0.001 is set to tolerance.
  - `-maximize` — If maximize option is given, sddsoptimize maximizes measurement by varying control correctors. Otherwise, it minimizes the measurement.
  - `-runControlPV` – specifies the runControl PV record.
  - `-runControlDescription` – specifies a string parameter whose value is a runControl PV description record.
  - `-testValues` – ¡filename¿ is name of an sdds format file containing minimum and maximum values of PV's specifying a range outside of which the feedback is temporarily suspended. Column names are ControlName, MinimumValue, MaximumValue. limit specifies the maximum times of testing when test fails.
- **see also:**
  - `sddsexperiment` (2.9)
- **author: H. Shang, ANL**

2.17 sddspvtest

- **description:** `sddspvtest` tests the process variable given by the inputfile are out-of-range or not and sets the number of out-of-range process variables to a control PV.

- **example:**

```
sddspvtest linac.sdds -time=20 -runControlPV={string=shang:ControlLawRC,pingTimeout=4}\
  -runControlDescription=string=hi
```

  where the contents of the file `linac.sdds` are

```
SDDS1
&description text="Namecapture BURT Request File", contents="BURT Request", &end
&column name=ControlName, type=string,  &end
&column name=MaximumValue, type=double,  &end
&column name=MinimumValue, type=double,  &end
&data mode=ascii, &end
! page number 1
linac
                  10
soliday:PM1:X:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM1:Y:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM2:X:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM2:Y:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM3:X:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM3:Y:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM4:X:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM4:Y:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM5:X:positionM  1.000000000000000e+00 -1.000000000000000e+00
soliday:PM5:Y:positionM  1.000000000000000e+00 -1.000000000000000e+00
.......
```

- **synopsis:**

```
usage: sddspvtest <inputFile> [-pvOutput=<pvName>]
  [-time=<timeToRun>,<timeUnits>] [-interval=<timeInterval>,<timeUnits>]
  [-runControlPV={string=<string>|parameter=<string>},pingTimeout=<value>,
  pingInterval=<value>]
  [-runControlDescription={string=<string>|parameter=<string>}]
  [-pendIOtime=<value>] [-verbose] [-testValues=<file>[,limit=<number>]]
```

- **files:**
  - **input file:**
    The variable input file is an SDDS file with one string column: ControlName, which is required and gives the list of control correctors (process variables or knobs). It also contains two double double columns: MaximumValue and MinimumValue, which are also required. seconds between two settings of the correctors.

- **switches:**
  - `-pvOutput` — optional. the output pv name for storing the testing results of PVs in the input. If it is not given,the results are printed out.
  - `-time` —required. Total time for testing process variable. Valid time units are seconds,minutes,hours, or days.
  - `-interval` —optional. Desired time interval for testing, the units are same as time.
  - `-runControlPV` — specifies the runControl PV record. string—parameter is required, pingInterval and pingTimeout are optional.
  - `-runControlDescription` —specifies a string parameter whose value is a runControl PV description record.
  - `-verbose` – print out messages.
  - `-pendIOtime` – sets the maximum time to wait for return of each value.
  - `-testValues<file>,[limit=<number>]` – file is sdds format file containing minimum and maximum values of PV's specifying a range outside of which the feedback is temporarily suspended. Column names are ControlName, MinimumValue, MaximumValue. Optional column names are SleepTime, ResetTime. limits is the maixum number of failure times. The program will be terminated when the continuous failure times reaches the limit.
- **see also:**
  - `sddscontrollaw` (2.8)
- **author: H. Shang, ANL**

2.18 sddssnapshot

- **description:** `sddssnapshot` reads values of process variables and writes them to a file. An input file lists the process variables to be read.

  `sddssnapshot` differs from `burtrb` in that `sddssnapshot` may operate in a server mode in which a new file is written to the named output file whenever the signal SIGUSR1 is received by `sddssnapshot`. Another improvement over `burtrb` is that all data in the input file (even those not needed by the program) are transfered to the output file. `sddssnapshot` is more for data collection as opposed to backup and restore.

- **example:** The state of the APS storage ring is saved by writing values of process variables listed in SR.req to the snapshot file SR.snp:

  ```
  sddssnapshot SR.req SR.snp -nameOfData="Value"
  ```

  where the contents of the file `SR.req` are

  ```
  SDDS1
  &column
   name = ControlName,  type = string, &end
  &data
   mode = ascii, &end
  &data
   mode = "ascii", no_row_counts=1 &end
  S1A:Q1:CurrentAO
  S1A:Q2:CurrentAO
  ...
  ```

- **synopsis:**

  ```
  usage: sddssnapshot [-pipe[=input][,output]] [<input>] [<output>]
  [-ezcaTiming=<timeout>,<retries>] [-unitsOfData=<string>] [-nameOfData=<string>]
  [-serverMode=<pidFile>] [-average=<number>,<intervalInSec>]
  Takes a snapshot of EPICS scalar process variables.
  Requires the column "ControlName" with the process variable names.
  For server mode, writes a new file to the given filename whenever
  SIGUSR1 is received.  Exits when SIGUSR2 is received.
  ```

- **files:**
  - **input file:**
    The input file is an SDDS file with at least one column:
    * `ControlName` — Required string column for the process variable or device name.
  - **output file:**
    The output file contains all columns of the input file including those not needed by the program plus a column named on the command line option `-nameOfData`. This column is defined as a double type and contains the readback values. Optionally the units of that readback column may be specified on the command line. Of course this option is useful

only if all the process variables have the same units, as in the case of recording orbit values from all bpms. If the `-average` option is requested, then an additional double column is created with the name of the readback column with the "StDev" appended to it.

  - **pid file:**

    A process id file is created with option `serverMode=<pidFile>`. This file contains a single number which is the pid number of the running `sddssnapshot` process.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-ezcaTiming=<timeout>,<retries>` — Specifies tuning of ezca, the channel access interface used by `sddssnapshot`.
  - `-nameOfData=<string>` — Column name to be given to the data collected. Default name is "Value".
  - `-unitsOfData=<string>` — Optional. Name given to the units field of the column definition of the data to be collected. Default value is the null string.
  - `-serverMode=<pidFile>` — Optional. Enables the server more. The file specified will be created and contain the process number of the present `sddssnapshot` process. This file is the mechanism through which the user will know to which process should the SIGUSR1 be sent. To activate one snapshot write, the user can type the command "`kill -SIGUSR1 'cat <pidFile>'`".
  - `-average=<number>,<intervalInSec>` — Optional. On can specify the number of readings to average and the number of seconds interval between readings.

- **see also:**
  - `burtrb` (2.1)
  - `sddsmonitor` (2.15)

- **author: M. Borland, ANL**

2.19 sddsstatmon

- **description:** `sddsstatmon` reads EPICS process variables, collects statistics, and writes these statistics to an output file. The statistics are the mean, standrd deviation, mininum, maximum, and sigma. An input file defines the process variables to be monitored.
- **example:** Statistics of the pressure readbacks of storage ring ion pumps and the stored current for groups of 60 data points taken at 1 second interval are collected with the command below.

```
sddsstatmon SRvac.mon SRvac.sdds -time=24,hours -interval=1,second \
 -samplesPerStatistic=60
```

  where the contents of the file `SRvac.mon` are

```
SDDS1
&description &end
&column
 name = ControlName,  type = string, &end
&column
 name = ControlType,  type = string, &end
&column
 name = ReadbackUnits,  type = string, &end
&column
 name = ReadbackName,  type = string, &end
&data
 mode = ascii, no_row_counts=1 &end
! page number 1
S35DCCT:currentCC pv mA S35DCCT
VM:01:3IP1.VAL pv Torr VM:01:3IP1
VM:01:2IP2.VAL pv Torr VM:01:2IP2
VM:01:2IP3.VAL pv Torr VM:01:2IP3
...
```

- **synopsis:**

```
usage: sddsstatmon <input> <output>
    [-erase | -generations[=digits=<integer>][,delimiter=<string>]]
    [-steps=<integer-value> | -time=<real-value>[,<time-units>]]
    [-interval=<real-value>[,<time-units>] | [-singleShot{=noprompt | stdout}]
    [-samplesPerStatistic=<integer>]
    [-verbose] [-precision={single|double}]
    [-updateInterval=<integer>]
    [-ezcaTiming[=<timeout>,<retries>]] [-noezca]
    [-oncaerror={skip | exit | repeat}
    [-comment=<parameterName>,<text>]
    [-getUnits={force | ifBlank | ifNoneGiven}]
    Writes values of process variables or devices to a binary SDDS file.
```

- **files:**
  - **input file:**
    The input file is an SDDS file with a few data columns required:
    * `ControlName` or `Device` — Required string column for the names of the process variables or devices to be monitored. Both column names are equivalent.
    * `Message` — Optional string column for the device read message. If a row entry in column `ControlName` is a process variable, then the corresponding entry in `Message` should be a null string.
    * `ReadbackName` — Optional string column for the names of the data columns in the output file. If absent, process variable or device name is used.
    * `ReadbackUnits` — Optional string column for the units fields of the data columns in the output file.If absent, units are null.
    * `ScaleFactor` — Optional double column for a factor with which to multiply values of the readback in the output file.
  - **output file:**
    The output file contains one column per statistic per process variable monitored. The five statistics are the mean, standard deviation, minimum value, maximum value, and sigma. The corresponding column names are `<name>Mean`, `<name>StDev`, `<name>Min`, `<name>Max`, and `<name>Sigma`, where `<name>` is the `ReadbackName` name value of the process variable in the input file.

    By default, the data type is float (single precision). Time columns and other miscellaneous columns are defined:
    * `Time` — Double column of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
    * `TimeOfDay` — Float column of system time in units of hours. The time does not wrap around at 24 hours.
    * `DayOfMonth` — Float column of system time in units of days. The day does not wrap around at the month boundary.
    * `Step` — Long column for step number.
    * `CAerrors` — Long column for number of channel access errors at each reading step.

    Many time-related parameters are defined in the output file:
    * `TimeStamp` — String parameter for time stamp for file.
    * `PageTimeStamp` — String parameter for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The `PageTimeStamp` value for the new page is the creation date of the new page. The `TimeStamp` value for the new page is the creation date of the very first page.
    * `StartTime` — Double parameter for start time from the `C` time call cast to type double.
    * `YearStartTime` — Double parameter for start time of present year from the `C` time call cast to type double.
    * `StartYear` — Short parameter for the year when the file was started.
    * `StartJulianDay` — Short parameter for the day when the file was started.

        * `StartMonth` — Short parameter for the month when the file was started.
        * `StartDayOfMonth` — Short parameter for the day of month when the file was started.
        * `StartHour` — Short parameter for the hour when the file was started.

- **switches:**
    - `-erase` — If the output file already exists, then it will be overwritten by `sddsstatmon`.
    - `-generations[=digits=<integer>][,delimiter=<string>]` — The output is sent to the file `<SDDSoutputfile>-<N>`, where `<N>` is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating `<N>`, so that the first generation number is 0001.
    - `-interval=<real-value>[,<time-units>]` — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
    - `-steps=<integer-value>` — Number of readbacks for each process variable before normal exiting.
    - `-time=<real-value>[,<time-units>]` — Total time for monitoring. Valid time units are seconds, minutes, hours, and days. The program calculates the number of steps by dividing this time by the interval. The completion time may be longer, because the time interval in not garanteed.
    - `-singleShot[=noprompt]` — a single read is prompted at the terminal and initiated by a `<cr>` key press. The time interval is disabled. With `noprompt` present, no prompt is written to the terminal, but a `<cr>` is still expected. Typing "q" or "Q" terminates the monitoring.
    - `-samplesPerStatistic=<integer>` — The number of samples to use for computing each statistic. The default is 25.
    - `-verbose` — Prints out a message when data is taken.
    - `-precision={single|double}` — Selects teh data type for the statistics columns.
    - `-updateInterval=<integer>` — Number of sample sets between each output file update. The default is 1.
    - `-ezcaTiming[=<timeout>,<retries>]` — Sets EZCA timeout and retry parameters.
    - `-noezca` — Obsolete.
    - `-oncaerror={usezero|skiprow|exit}` — Selects action taken when a channel access error occurs. The default is using zero (`usezero`) for the value of the process variable with the channel access error, and resuming execution. The second option (`skiprow`) is to

- **see also:**
    - `sddsvmonitor` (2.22)
    - `sddswmonitor` (2.23)
    - `sddssnapshot` (2.18)

- **author: M. Borland, ANL**

2.20 sddssynchlog

- **description:** `sddssynchlog` Reads values of process variables synchronously and writes them to an output file. Synchronism is imposed by requiring that all PVs generated callbacks occur within the number of seconds given by the -synchInterval argument.

- **example 1:**

  The linac bpms and MV200 PVs are synchonously monitored using the command below. One might wish to monitor these PVs determine if linac beam position and transverse size at a flag are correlated. The output file will contain 100 aligned samples of the PV data and should therefore take 16.67 seconds assuming the linac is running at 6 Hz. Alignment of process variable data is accepted and written to the output file if and only if the PV time stamps are withing +/- 1/6 Hz = 0.08 seconds. Data acquisition stops after 100 samples are acquired or a -timelimit of 20 seconds has elapsed. Note, it is absolutely required that the scalar synchronous PV file listed below (synchPVs.mon) contain at least a single PV. This is true even if one wishes only to synchronously acquire waveform process variables.

  ```
  sddssynchlog synchPVs.mon synchPVs.sdds -samples=100 -synchInterval=0.08
  -timelimit=20,seconds -verbose
  ```

  where the contents of the file `synchPVs.mon` are

  ```
  SDDS1
  &description &end
  &column
   name = ControlName,  type = string, &end
  &column
   name = ControlType,  type = string, &end
  &column
   name = ReadbackUnits,  type = string, &end
  &column
   name = ReadbackName,  type = string, &end
  &data
   mode = ascii, no_row_counts=1 &end
  ! page number 1
  L2:PM1:BPM.CX  pv mm L2:PM1:BPM.XPOS
  L2:PM1:BPM.CY  pv mm L2:PM1:BPM.YPOS
  L2:PM2:BPM.CX  pv mm L2:PM2:BPM.XPOS
  L2:PM2:BPM.CY  pv mm L2:PM2:BPM.YPOS
  LI:VD1:x:raw:cal:sigmaM pv Pixels LVid:xRawSigma
  LI:VD1:y:raw:cal:sigmaM pv Pixels LVid:yRawSigma
  ...
  ```

- **example 2:**

  The scalar linac bpm and MV200 PVs are synchonously monitored along with LCLS bpm data waveforms using the command below. The command is identical to that in example 1 except for the additional waveform file command which that specifies a file containing waveform PV data.

```
sddssynchlog synchPVs.mon synchPVs.sdds -waveformData=LCLS.wmon
-samples=100 -synchInterval=0.08 -timelimit=20,seconds -verbose
```

where the contents of the file `LCLS.wmon` are

```
SDDS1
&parameter name=WaveformLength, type=long, &end
&column
 name=WaveformPV, type=string,  &end
&column
 name=WaveformName, type=string,  &end
&data mode=ascii, &end
! page number 1
128
                   9
PAD:K211:1:CH0_RAW_WF PAD:K211:1:CH0_RAW_WF
PAD:K211:1:CH1_RAW_WF PAD:K211:1:CH1_RAW_WF
PAD:K211:1:CH2_RAW_WF PAD:K211:1:CH2_RAW_WF
PAD:K211:2:CH0_RAW_WF PAD:K211:2:CH0_RAW_WF
PAD:K211:2:CH1_RAW_WF PAD:K211:2:CH1_RAW_WF
PAD:K211:2:CH2_RAW_WF PAD:K211:2:CH2_RAW_WF
PAD:K211:3:CH0_RAW_WF PAD:K211:3:CH0_RAW_WF
PAD:K211:3:CH1_RAW_WF PAD:K211:3:CH1_RAW_WF
PAD:K211:3:CH2_RAW_WF PAD:K211:3:CH2_RAW_WF
...
```

- **synopsis:**

```
sddssynchlog <input> <output> [-slowData=<input>]
[-waveformData=<filename>] -samples=<number> [-timeLimit=<value>[,<units>]]
-eraseFile [-pendIOTime=<seconds>] [-connectTimeout=<seconds>]
[-verbose] [-comment=<parameterName>,<text>]
[-synchInterval=<seconds>] [-precision={single|double}]
[-saveTimeStamps=<filename>] [-steps=<int>] [-interval=<value>[,<units>]]

Logs numerical data for the process variables named in the ControlName
column of <input> to an SDDS file <output>.  Synchronism is imposed by
requiring that all PVs generated callbacks within the number of seconds
given by the -synchInterval argument.
```

- **files:**
  - **input file:**
    The input file is an SDDS file with a few data columns required:
    * `ControlName` or `Device` — Required string column for the names of the process variables or devices to be monitored. Both column names are equivalent.
    * `Message` — Optional string column for the device read message. If a row entry in column `ControlName` is a process variable, then the corresponding entry in `Message` should be a null string.

* `ReadbackName` — Optional string column for the names of the data columns in the output file. If absent, process variable or device name is used.
* `ReadbackUnits` — Optional string column for the units fields of the data columns in the output file.If absent, units are null.
* `ScaleFactor` — Optional double column for a factor with which to multiply values of the readback in the output file.

– **waveformData file:**

The waveformData file is an optional input file specified on the command line which lists waveform PVs to be synchronously logged along with the scalar PVs in the input file.

* `WaveformPV` — Required string column containing the names of the waveform process variables to be monitored.
* `WaveformName` — Optional string column for the names of the waveform data columns in the output file.
* `WaveformLength` — Required long parameter giving the number of waveform elements to be acquired. It can be less than the maximum number of waveform elements.

– **slowData file:**

The slowData file is an optional input file specified on the command line which lists scalar PVs to be slow (read non-synchronously) logged along with the synchronously logged scalar or waveform PVs in the input and waveform PV files. This feature is convenient if one wishes to log PVs for system information purposes instead of correlation analysis (such as beam current, vacuum pressure etc.). This file has the same columns as the input file.

– **output file:**

The output file contains one data column for each process variables named in the input file if no waveformData file is specified. By default, the data type is float (single precision). If an optional waveformData file is specified, the structure of the output file changes. In this case, the column data are the waveform PVs and the scalar PVs in the input and slowData files are parameters. Each page contains waveform and scalar PV data for a single sample of the total number of samples acquired. The total number of pages equal the number of samples specified in -samples if all samples are able to be collected (ie. timeLimit is not exceeded during acquisition). Time columns and other miscellaneous columns are defined:

* `Time` — Double column of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
* `Sample` — Long column for sample number.

Many time-related parameters are defined in the output file:

* `TimeStamp` — String parameter for time stamp for file.
* `PageTimeStamp` — String parameter for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The `PageTimeStamp` value for the new page is the creation date of the new page. The `TimeStamp` value for the new page is the creation date of the very first page.
* `StartTime` — Double parameter for start time from the `C` time call cast to type double.

    * YearStartTime — Double parameter for start time of present year from the C time call cast to type double.
    * StartYear — Short parameter for the year when the file was started.
    * StartJulianDay — Short parameter for the day when the file was started.
    * StartMonth — Short parameter for the month when the file was started.
    * StartDayOfMonth — Short parameter for the day of month when the file was started.
    * StartHour — Short parameter for the hour when the file was started.

- **switches:**
  - -samples — Specifies the number of samples to attempt. If synchronism problems are encountered, fewer samples will appear in the output.
  - -timeLimit — Specifies maximum time to wait for data collection.
  - -eraseFile — Specifies erasing the file ¡output¿ if it exists already.
  - -pendIOtime=<value> — Obsolete.
  - -connectTimeout=<value> — Specifies maximum time in seconds to wait for a connection before issuing an error message. 60 is the default.
  - -verbose — Specifies printing of possibly useful data to the standard output.
  - -comment=<parameterName>,<text> — Gives the parameter name for a comment to be placed in the SDDS output file, along with the text to be placed in the file.
  - -synchInterval — Specifies the time spread allowed for callbacks from the PVs being logged. If any PV fails to callback within the specified interval, the data for that sample is discarded.
  - -slowData — Specifies an sddsmonitor-type input file giving the names of PVs to be acquired at a "slow" rate, i.e., without synchronization. Data for these PVs are interpolated linearly to get values at each time sample of the synchronous data.
  - -waveformData — Specifies an sddswmonitor-type input file giving the names of waveform PVs to be synchronously logged. If this option is given, the output file is changed so that waveform data is stored in columns while scalar data is stored in parameters.
  - -saveTimeStamps — Specifies the name of a file to which to write raw time-stamp data for the synchronously-acquired channels. This can be useful in diagnosing the source of poor sample alignment.
  - -precision Specifies PV data to be either single or double precision. Default is double precision.
  - -steps specifies how many synchlogs will be taken with -samples of data, default is 1.
  - -interval specifies the waiting time between two synchlog steps.

- **see also:**
  - sddsmonitor (2.15)
  - sddswmonitor (2.23)
  - sddsvmonitor (2.22)

- **author: M. Borland, ANL**

2.21 sddsvexperiment

- **description:** `sddsvexperiment` varies process variables and measures process variables, with optional averaging. An input file of namelist commands gives the specific instructions. The results are recorded in one or more SDDS files. This command differs from `sddsexperiment` in that the measurement process variables are specified in two lists. The first list gives PV "rootnames". The second list gives suffixes to apply to each of the rootnames.

- **example:** The strength of a storage ring horizontal corrector (S1A:H1) is varied while the readbacks at all horizontal beam position monitors are recorded. The output file is S1A:H1.sdds.

  ```
  sddsvexperiment S1A:H1.vexp S1A:H1.sdds
  ```

  where the contents of the file `S1A:H1.exp` are

  ```
  &rootname_list
          filename=bpmRoot.sdds
  &end
  &suffix_list
          filename=bpmSuffix.sdds
  &end

  &variable PV_name = "S1A:H1:CurrentAO",
          parameter_name="S1A:H1"
  ! the corrector is varied in 5 steps from -1.0 to 1.0 amps.
          index_number = 0, index_limit = 5,
          initial_value = -1.0, final_value = 1.0,
  &end

  &execute
          post_change_pause=4,
          intermeasurement_pause=1
  &end
  ```

  where the line starting with a "!" is a comment.

  The contents of the file `bpmRoot.sdds` is

  ```
  SDDS1
  &description &end
  &column
   name = "Rootname",  type = "string", &end
  &data
   mode = "ascii", &end
  360     ! number of rows
  S1A:P1
  S1A:P2
  S1A:P3
  S1A:P4
  ```

```
S1B:P5
S1B:P4
S1B:P3
S1B:P2
S1B:P1
S2A:P1
...
```

The contents of the file `bpmSuffix.sdds` is

```
SDDS1
&description &end
&column name=Suffix, type=string &end
&column name=NumberToAverage, type=long &end
&data mode=ascii, no_row_counts=1 &end
:ms:x   10
:ms:y   10
```

- **synopsis:**

```
usage: sddsvexperiment  <inputFile> <outputFile>
[-suffixFile=<filename>] [-rootnameFile=<filename>]
[-echoinput] [-dryrun] [-summarize] [-verbose[=very]]
[-ezcaTiming=<timeout>,<retries>] [-describeInput]
```

- **files:**
  - **input file:**
    The input file consists of namelist commands that set up and execute the experiment. The functions of the commands are described below.
    * `variable` — specifies a process variable to vary, and the range and steps of the variation. More than one variable command may be defined, so that many process variables may vary at a time.
    * `rootname_list` — specifies rootnames from which to generate process variable to measure at each step during the experiment.
    * `suffix_list` — specifies suffixes from which to generate process variable to measure at each step during the experiment.
    * `execute` — start executing the experiment. One group of variable, measurement and execute commands may follow another in the same file for multiple experiments.
    * `erase` — deletes previous variable or measurement setups.
    * `list_control_quantities` — makes a cross-reference file for process variable names and column names of the data file.
    * `system_call` — specifies a system call (usually a script) to be executed either before a measurement or before setting a process variable.

    The following text describes all the namelist commands and their respective fields in more detail. The command definition listing is of the form

```
&<command-name>
    <variable-type> <variable-name> = <default-value>
    .
    .
    .
&end
```

where the part `<variable-type>`, which doesn't appear in an actual command, is used to illustrate the valid type of the value. The three valid types are:

* `double` — for a double-precision type, used for most physical quantity variables,
* `long` — for an integer type, used for flags mostly.
* `STRING` — for a character string enclosed in double quotes.

An actual namelist in an input file should look like this:

```
&<command-name>
    [<variable-name> = <value>,]
    ...
&end
```

In the namelist definition listings the square brackets denotes an optional component. Not all variables need to be defined – the defaults may be sufficient. Those that do need to be defined are noted in the detailed explanations. The only variables that don't have default values in general are string variables.

variable

* function: Specifies a process variable to vary, and the range and steps of the variation. Values of variables at each measurement step are written to an SDDS output file. The readback-related fields are used to confirm that the physical device has responded to a setpoint command at every step (and substep) within some tolerance. Readback is enabled when `readback_attempts` and `readback_tolerance` are defined with non-zero positive values.
  When an arbitrary sequence of setpoint values is required (say a binary sequence), the values can be read in from an SDDS file specified by the `values_file` field. The fields associated for the range and steps are ignored in this case.
  With multiple `variable` commands, variables may be varied in a multi-dimensional grid. For example, variables may be varied independently of each other, or some groups of variables may vary together forming one axis of a multi-dimensional grid (see item `index_number`).

```
&variable
        STRING PV_name = NULL
        STRING parameter_name = NULL
        STRING symbol = NULL
        STRING units = "unknown"
        double initial_value = 0
        double final_value = 0
        long relative_to_original = 0
        long index_limit = 0
        long index_number = 0
        STRING function = NULL
        STRING values_file = NULL;
        STRING values_file_column = NULL;
        long substeps = 1
        double substep_pause = 0
        double range_multiplier = 1
        STRING readback_name = NULL
        double readback_pause = 0.1
        double readback_tolerance = 0
        long readback_attempts = 10
        long reset_to_original = 1
&end
```

* `PV_name` — Required. Process variable name to vary.
* `parameter_name` — Required. Parameter name for the variable data recorded in the output file.
* `symbol` — Optional. Symbol field for the above column definition of the variable data.
* `units` — Optional. Units field for the above column definition of the variable data.
* `initial_value` — Required. The initial value of the process variable in the variation.
* `final_value` — Required. The final value of the process variable in the variation.
* `index_limit` — Number of steps in the variation. Measurements are taken at each step.

* index_number — Required. The counter (or index) number with which the defined variation is associated. In a sddsexperiment run, counters must be defined in an increasing sequence starting from counter 0. That is, the first variable command of the file must have index_number = 0. The second variable command must have index_number = 0 or 1. In the former case, the two variables will move together with the same number of steps according their respective initial_value and final_value. In the latter case, the two variables will vary independently of each other with possibly different number of steps in a 2-dimensional grid.
  Counter number n is nested within counter $n+1$. Therefore it might be efficient to assign devices with slower response times to higher index_number counter.
* index_limit — Normally required. Number of steps in the variation. Measurements are taken at each step. When more than one variable is associated with the same counter, only the index_limit of the first variable definition for that counter need to be defined. If index_limit is defined in variable commands of the same index_number value, then the first index_limit remain in force.
* relative_to_original — Optional. If non-zero, then the variation range is defined relative to the original process variable value (i.e. the value prior to running the program).
* range_multiplier — Optional. Factor by which the range, final_value - initial_value, is multiplied. New values of initial_value and final_value are calculated while keeping the midpoint of the range the same.
* function — Optional. A string of rpn operations used to transform the range specified by initial_value, final_value, and index_limit. For convenience, the original value of the process variable, and the calculated grid value for the process variable on the current step or substep are automatically pushed onto the the stack before the function is executed. The calculated values are recorded in the output file. The environment variable RPN_DEFNS is used to read a rpn definition file at the start of the execution of sddsvexperiment.
* values_file — Optional. An SDDS data file containing setpoints for the variable. This is useful is one has arbitrary setpoints values to apply. The values of the fields initial_value, final_value, _substeps, range_multiplier and index_limit are ignored.
  One can have other variable namelists with the same index_number that don't use a file for the values. The default index_limit of the other variable will be set to the number of setpoint in the values file. Thus the values in the file and the values calculated for the other variable will vary together with the same number of steps.
* values_file_column — Required when values_file is specified. values_file_column gives the column name of the setpoints data in file values_file.
* substeps — Optional. If greater than one, the steps are subdivided into this number. Measurements are not made at substeps. Substeps are useful when the physical device associated with the process variable cannot reliably make steps as large as those that might be defined with initial_value, final_value, and index_limit.
* substep_pause — Optional. Number of seconds to pause after the variable change of each substeps.
* readback_name — Optional. Readback process variable name associated with PV_name. The default value for readback_name is PV_name.

* `readback_tolerance` — Optional. Maximum acceptable absolute value of the difference between the process variable setpoint and its readback. A positive value is required in order to enable readbacks.
* `readback_pause` — Optional. Number of seconds to pause after each reading of the `readback_name` process variable. This pause time is in addition to other pauses defined.
* `readback_attempts` — Optional. Number of allowed readings of the `readback_name` process variable and readback pauses after a variable change has occured. After this number of readings, the program exits. The first readback is attempted immediately (i.e. no pause) after sending a setpoint command to the `PV_name`. A positive value is required in order to enable readbacks.
* `reset_to_original` — Optional. A value of 1 means that the variable is reset to its original value when the experiment terminates normally or abnormally.

rootname_list

* function: specifies rootnames from which to generate names of process variable to measure at each step during the experiment.

```
&rootname_list
        STRING filename = NULL;
&end
```

* `filename` — Required. See description of rootname file below.

suffix_list

* function: specifies suffixes from which to generate names of process variable to measure at each step during the experiment.

```
&suffix_list
        STRING filename = NULL;
&end
```

* `filename` — Required. See description of suffix file below.

execute

* function: start executing the experiment. Some global parameters are defined here.

```
&execute
        double post_change_pause = 0
        double intermeasurement_pause = 0
        double rollover_pause = 0
        long post_change_key_wait = 0
        long allowed_timeout_errors = 1
        long allowed_limit_errors = 1
        double outlimit_pause = 0.1
        long repeat_reading = 1
        double post_reading_pause = 0.1
        double ramp_pause = 0.25;
        long ramp_steps = 10;
&end
```

* `post_change_pause` — Optional. Number of seconds to pause after each change before attempting to make any measurement.
* `intermeasurement_pause` — Optional. Number of seconds to pause between each measurement. Individual measurements for averaging are taken at this interval.
* `rollover_pause` — Optional. Number of seconds to pause after a counter has reached its upper limit, and must rollover to zero. This allows any physical devices associated with the counter to settle after a change equal to the total range of the variation.
* `post_change_key_wait` — Optional. If non-zero, then wait for a key press after making variable changes but before taking measurements. A prompt is given.
* `allowed_timeout_errors` — Optional. Number of timeout errors allowed before aborting the program.
* `allowed_limit_errors` — Optional. Number of invalid range measurement errors allowed before aborting the program. The valid range of a measurement is specified in the `measurement` command.
* `outlimit_pause` — Optional. Number of seconds to pause after an invalid range measurement error occured. This is to permit equipment time to recover from whatever glitch caused the out-of-limit reading.
* `repeat_reading` — Optional. The measurements and statistical analyses are repeated this number of times for each variable settings. A page of data is written to the output file for each repitition.
* `post_reading_pause` — Optional. Number of seconds to pause after taking a set of measurements and making a statistical analysis. If measurements are repeated then the pause is repeated after each set of measurements.
* `ramp_steps` — Optional. Number of steps in the variables PV ramp which occurs at the start and the end of the experiment.
  Ramping is necessary for some devices that do not respond well to large changes to their setpoints. Ramping is done at the start of the experiments to slowly change the variable PVs from their current values to their initial values. Another ramp is done at the end to slowly bring the variable PVs from their final values back the

original values. Ramping back to original values is also done when the experiment aborts for some reason.

* ramp_pause — Optional. Time interval at each step of the variables PV ramp which occurs at the start and the end of the experiment. This is not the same variable as the pause between variable changes during the experiment.

erase

* function: deletes previous variable or measurement setups.

```
&erase
        long variable_definitions = 1
        long measurement_definitions = 1
&end
```

* `variable_definitions` — Optional. If non-null, then all the variable definitions are erased.
* `measurement_definitions` — Optional. If non-null, then all the measurement definitions are erased.

list_control_quantities

* function: makes a cross-reference file for process variable names and column names of the data file.

```
&list_control_quantities
         STRING filename = NULL
&end
```

* `filename` — Required. Name of file. Columns defined are `ControlName`, `SymbolicName`, and `ControlUnits`.

system_call

* function: specifies a system call (usually a script) to be executed repeatedly during the experiment.

```
&system_call
        STRING command = NULL
        long index_number = 0
        long index_limit = 0
        double post_command_pause = 0
        double pre_command_pause = 0
        long append_counter = 0
        STRING counter_format = "%ld"
        long call_before_setting = 0
        long call_before_measuring = 1
        STRING counter_column_name = NULL
&end
```

* `command` — Required. Name of shell command or script to execute.
* `index_number` — Required. Counter number with which the command will be associated. The command is executed when this counter is advanced or rolled over.
* `index_number` — Optional. Number of times the command is executed for the associated counter. This field is used only when the value of `index_number` above defines a new counter.
* `post_command_pause` — Optional. Number of seconds to pause after the completion of the command.
* `pre_command_pause` — Optional. Number of seconds to pause before executing the command.
* `append_counter` — Optional. If non-zero, the counter value is appended to the command when the system call is made.
* `counter_format` — Optional. Format for the counter if the counter value is appended to the command.
* `call_before_setting`, `call_before_measuring`, — Optional. At a counter advance or rollover the command can be executed in one of three ways:
  · before both variable changes and measurements:
  `call_before_setting`=1, `call_before_measuring`=1
  · after variable changes and before measurements:
  `call_before_setting`=0, `call_before_measuring`=1
  · after both variable changes and measurements:
  `call_before_setting`=0, `call_before_measuring`=0

  If multiple measurements are made for averaging, the command is not executed between measurements.
* `counter_column_name` — Optional. If non-null, a column in the output file with this name is defined. The values written to this column are the number of times the command had been called minus one. This value doesn't rollover with its associated counter.

– **Rootname file:**

SDDS file defining the rootnames of the process variables with colum:

  * `Rootname` — Required string column to generate measurement process variables.

– **Suffix file:**

SDDS file defining the suffixes and measurement parameters with columns:

  * `Suffix` — Required string columne of suffix names to be appended to the `Rootname` values in the rootname file to generate measurement process variables.
  * `ColumnNameSuffix` — Optional string column of names of suffix column appearing in output data file.
  * `NumberToAverage` — Optional long column of number of measurements to average.
  * `IncludeStDev` — Optional character column. If value is "y" then the standard deviation (a measure of the distribution of measurements) is calcualted and included in the output file. If "n" then the standard deviation is not calculated.
  * `IncludeSigma` — Optional character column. If value is "y" then the sigma (uncertainty on the mean value) is calcualted and included in the output file. If "n" then the sigma is not calculated.
  * `LowerLimit` and `UpperLimit` — Optional double columns. Must have both or neither. Defines a range of validity for the individual measurements. If the number of invalid measurements (reset to 0 at each measurement step) equals or exceeds the value of `allowed_limit_errors` (default of 1) in command `execute`, then the program aborts. The average values written to the output file excludes measurements outside this range.

– **Output file:**

The output file contains one data page for each variable step. The names of the defined columns are those string data of the `Suffix` or `ColumnNameSuffix` columns from the suffix file. A column is created for each standard deviation or sigma calculation requested for a measurement. The standard deviation columns are named `StDev<columnName>`, and the sigma columns are named `StDev<columnName>`, where `<columnName>` is replaced by an actual column name.

Some additional columns are defined:

  * `Rootname` — String column for the rootname of the PVs.
  * `Index` — Long columns for the index of the row.
  * `Time` — Double parameter of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
  * `ElapsedTime` — Double parameter of elapsed time of readback since the start of the experiment.

The variable values appear as parameters in each data page.

Many time parameters are defined:

  * `Step` — Long parameter for the step number.
  * `Time` — Double parameter of time since start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.

    * `ElapsedTime` — Double parameter of elapsed time of readback since the start of the experiment.
    * `TimeOfDay` — Double parameter of system time in units of hours. The time does not wrap around at 24 hours.
    * `TimeStamp` — String parameter of time stamp for file.

- **switches:**
  - `-suffixFile=<filename>` — SDDS file defining the suffixes and measurement parameters. If not specified on the command line, then the namelist command `suffix_list` is required in the input file.
  - `-rootnameFile=<filename>` — SDDS file defining the rootnames of the process variables. If not specified on the command line, then the namelist command `rootname_list` is required in the input file.
  - `-echoinput` — echos input file to stdout.
  - `-dryrun` — the "variable" process variables are left untouched during the execution. The "measurement" process variables are still read. The pauses and system calls are still in effect.
  - `-summarize` — gives a summary of the experiment before executing it.
  - `-verbose[=very]` — prints out information during the execution such as notification of setting and reading process variables. The option `very` prints out the average measurement values.
  - `-ezcaTiming=<timeout>,<retries>` — sets EZCA timeout and retry parameters
  - `-describeinput` — Printouts the list of namelist commands and fields of the input file.
- **see also:**
  - `sddsexperiment` (2.9)
- **author: M. Borland, ANL**

2.22 sddsvmonitor

- **description:** `sddsvmonitor` reads values of process variables and writes them to a file at a specified time interval. This command differs from `sddsmonitor` in that the monitored process variables names are specified in two lists. The first list gives PV "rootnames". The second list gives suffixes to apply to each of the rootnames. For each readback step, a page is written to the output file with the PV rootnames appearing in one column, and the process variable values in separate data columns for each suffix.

  Warning: If the readback values of all of the vector PVs do not change, then no data sets are written to the output file. This skipping of duplicate values is intended to keep the size of the output file as small as possible. The scalar PVs are not checked for changes though. In the future an option that allows logging of duplicate vector PVs may be implemented.

- **example:** The pressures readbacks of storage ring vacuum gauges are monitored with the command below.

  ```
  sddsvmonitor SRvac.vmon SRvac.vsdds -time=24,hours -interval=1,minute
  ```

  where the contents of the file `SRvac.vmon` are

  ```
  SDDS1
  &parameter name=ListType, type=string &end
  &column
   name = "ListData",  type = "string", &end
  &data
   mode = "ascii", no_row_counts=1 &end
  Rootnames
  VM:01:
  VM:02:
  VM:03:
  ...
  VM:40:

  Suffixes
  VGC1.PRES
  ```

  There is only one element in the suffix list of this example. The output file will contain columns `Rootnames` and `VGC1.PRES`.

- **synopsis:**

  ```
  usage: sddsvmonitor {<inputfile> | -rootnames=<file> -suffixes=<file>]
      [-scalars=<filename>] <outputfile>
      [{-erase | -append | -generations[=digits=<integer>][,delimiter=<string>}]
      [-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]]
      [-steps=<integer> | -time=<value>[,<units>]] [-interval=<value>[,<units>]]
      [-verbose] [-singleShot[=noprompt]] [-precision={single | double}]
      [-onCAerror={useZero | skipPage | exit}] [-PVlist=<filename>]
      [-noEzca [-pendIOtime=<value>]] [-ezcaTime=<timeout>,<retries>]
  ```

```
    [-logDuplicates[=countThreshold=<number>]]
    [-comment=<parameterName>,<text>]
Writes values of process variables to a binary SDDS file.
```

- **files:**
  - **input file:**

    The input file is an SDDS file with one data column and one parameter:

    * `ListData` — Required string column for the root part or the suffix part of the process variable names.
    * `ListType` — Required string parameter for the name part type. The only values recognized by sddsvmonitor are "`rootnames`" and "`suffixes`".

    The list of process variables is formed by combining all the rootnames and suffixes.
  - **rootname and suffix files:**

    An alternative to specifying the rootnames and suffixes with the above input file is to specify the list of rootnames and suffixes with two separate files, as shown in the usage message above. The string data in rootname file must be in column `Rootname`. The string data in suffix file must be in column `Suffix`.
  - **scalar PV input file:**

    An optional input file for scalar PVs (i.e. regular PVs) can be specified. The required columns are:

    * `ControlName` — Required string column for the names of the scalar process variables to be monitored.
    * `ReadbackName` — Required string column for the names of the parameter in the output file in which the values of the scalar process variables are written.
  - **conditions file:**

    The conditions file is an optional input specified on the command line which lists conditions that must be satisfied at each time steps before the data can be logged.

    The file is like the main input file, but has numerical columns `LowerLimit` and `UpperLimit`. The minimal column set is `ControlName`, which contain the PV names, and the two limits columns above. Depending on comand line options, when any or all PV readback from this file is outstide the range defined by the corresponding data from `LowerLimit` and `UpperLimit`, none of the data of the input file PVs are recorded. When this situations occurs for a long period of time, the size of the output file doesn't grow, and it may appear that the monitoring process has somehow stopped. It is possible to check the program activity with the `touch` sub-option which causes the monitoring program to touch the output file at every step.
  - **output file:**

    The output file contains one data column for each suffix named in the input file. By default, the data type is float (single precision). Other columns are:

    * `Index` — Long column for index of rootname.
    * `Rootname` — String column for rootnames from the input file.

    Each reading step produces a new page in the output file. Time and other miscellaneous parameters are defined:

    * `Time` — Double column for time of readback since the start of epoch. This time data can be used by the plotting program `sddsplot` to make the best coice of time unit conversions for time axis labeling.
    * `TimeOfDay` — Float column for system time in units of hours. The time does not wrap around at 24 hours.
    * `DayOfMonth` — Float column for system time in units of days. The day does not wrap around at the month boundary.
    * `Step` — Long column for step number.
    * `CAerrors` — Long column for number of channel access errors at each reading step.

  For each scalar PV defined in the `scalars` command line option a parameter of type double is defined.

  Many time-related parameters which don't change values throughout the file are defined:

    * `TimeStamp` — String column for time stamp for file.
    * `PageTimeStamp` — String column for time stamp for each page. When data is appended to an existing file, the new data is written to a new page. The `PageTimeStamp` value for the new page is the creation date of the new page. The `TimeStamp` value for the new page is the creation date of the very first page.
    * `StartTime` — Double column for start time from `C` time call cast to type double.
    * `YearStartTime` — Double column for start time of present year from `C` time call cast to type double.
    * `StartYear` — Short parameter for the year when the file was started.
    * `StartJulianDay` — Short parameter for the day when the file was started.
    * `StartMonth` — Short parameter for the month when the file was started.
    * `StartDayOfMonth` — Short parameter for the day of month when the file was started.
    * `StartHour` — Short parameter for the hour when the file was started.

- **switches:**
  - `-rootnames=<file>` — Specifies input file for rootnames. String values must be in column `Rootnames`.
  - `-suffixes=<file>` — Specifies input file for suffixes. String values must be in column `Suffixes`.
  - `-scalars=<filename>` — Specifies input file for scalar PV names. The values are logged as parameters.
  - `-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]` — Names an SDDS file containing PVs to read and limits on each PV that must be satisfied for data to be taken and logged. The file is like the main input file, but has numerical columns LowerLimit and UpperLimit.

    One of `allMustPass` or `oneMustPass` must be specified. It would make sense to use `allMustPass` in most monitoring applications. If `touchOutput` is present, then the output file is touched, even if no data is written. This way, one can determine by the time stamp of the file whether the monitoring job is still alive when the conditions fail for a long period of time. If `retakeStep` is present, then the value of `Step` in the output file is not incremented until the conditions pass, and data is written to the output file.
  - `-erase` — If the output file already exists, the it will be overwritten by sddsvmonitor.

- -append[=recover] — If the output file already exists, then append the new readings. The output file must have previously been generated by sddsvmonitor using the same information in the input files. The recover option allows an attempt to recover the data using sddsconvert if the input file is somehow corrupted.
- -generations[=digits=<integer>][,delimiter=<string>] — The output is sent to the file <SDDSoutputfile>-<N>, where <N> is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating <N>, so that the first generation number is 0001.
- -interval=<real-value>[,<time-units>] — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. Because the calls to the control system make take up a significant amount of time, the average effective time interval may sometimes be longer specified.
- -steps=<integer-value> — Number of readbacks for each process variable before normal exiting.
- -time=<real-value>[,<time-units>] — Total time for monitoring. Valid time units are seconds, minutes, hours, and days. The program calculates the number of steps by dividing this time by the interval. The completion time may be longer, because the time interval in not garanteed.
- -verbose — prints out a message when data is taken.
- -singleShot[=noprompt] — a single read is prompted at the terminal and initiated by a <cr> key press. The time interval is disabled. With noprompt present, no prompt is written to the terminal, but a <cr> is still expected. Typing "q" or "Q" terminates the monitoring.
- -oncaerror={usezero|skiprow|exit} — Selects action taken when a channel access error occurs. The default is using zero (usezero) for the value of the process variable with the channel access error, and resuming execution. The second option (skiprow) is to simply throw away all the data for that read step, and resume execution. the third option is to exit the program.
- -PVlist=<filename> — Specifies a file in which to write the names of all PVs monitored.
- -ezcaTiming[=<timeout>,<retries>] — Sets EZCA timeout and retry parameters.
- -logDuplicates[=countThreshold=<number>] — Specifies that data should be logged even if it is exactly the same as the last data.
- -comment=<parameterName>,<text> — Gives the parameter name for a comment to be placed in the SDDS output file, along with the text to be placed in the file.

- **see also:**
  - sddsvmonitor (2.22)
  - sddswmonitor (2.23)
  - sddssnapshot (2.18)
- **author: M. Borland and L. Emery, ANL**

2.23 sddswmonitor

- **description:** `sddswmonitor` reads values of waveform process variables and writes them to a file at a specified time interval. An input file defines the process variables to be monitored.

  Warning: If the readback values of all of the waveform PVs do not change, then no data sets are written to the output file. This skipping of duplicate values is intended to keep the size of the output file as small as possible. The scalar PVs are not checked for changes though. In the future an option that allows logging of duplicate waveform PVs may be implemented.

- **example:** The history of a beam position monitor readback is collected with this command:

  ```
  sddswmonitor SlowBh.wmon SlowBh.sdds -step=1
  ```

  where the contents of the file `SlowBh.wmon` are

  ```
  SDDS1
  &description &end
  &description
   contents = "sddssequence output", &end
  &parameter
   name = WaveformLength, type=long, &end
  &column
   name = WaveformPV,  type = string, &end
  &column
   name = WaveformName,  type = string, &end
  &data
   mode = ascii, &end
  ! page number 1
  512             ! WaveformLength
  2               ! number of rows
  S1A:P1:bh:x_wf S1A:P1:x
  S1A:P1:bh:y_wf S1A:P1:y
  ```

- **synopsis:**

  ```
  usage: sddswmonitor {<inputfile> | -PVnames=<name>[,<name>]} <outputfile>
      [{-erase | -generations[=digits=<integer>][,delimiter=<string>]}]
      [-steps=<integer> | -time=<value>[,<units>]] [-interval=<value>[,<units>]]
      [-verbose] [-singleShot[=noprompt]] [-precision={single | double}]
      [-onCAerror={useZero | skipPage | exit}]
      [-scalars=<filename>]
      [-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]]
      [-ezcaTime=<timeout>,<retries>]
      [-comment=<parameterName>,<text>]
  Writes values of waveform process variables to a binary SDDS file.
  ```

- **files:**
  - **input file:**
    The input file is an SDDS file with two required columns and one required parameter:

* `WaveformLength` — Required long parameter for the length of the waveform PV's. All WaveformPVs are expected to have this length.
* `WaveformPV` — Required string column for the names of the waveform process variables.
* `WaveformName` — Required string column for the names of the data columns in the output file.

– **scalar PV input file:**

An optional input file for scalar PVs (i.e. regular PVs) can be specified. The required columns are:

* `ControlName` — Required string column for the names of the scalar process variables to be monitored.
* `ReadbackName` — Required string column for the names of the parameter in the output file in which the values of the scalar process variables are written.

– **conditions file:**

The conditions file is an optional input file specified on the command line which lists conditions that must be satisfied at each time step before the data can be logged.

The file is like the main input file, but has numerical columns `LowerLimit` and `UpperLimit`. The minimal column set is `ControlName`, which contain the PV names, and the two limits columns above. Depending on comand line options, when any or all PV readback from this file is outstide the range defined by the corresponding data from `LowerLimit` and `UpperLimit`, none of the data of the input file PVs are recorded. When this situations occurs for a long period of time, the size of the output file doesn't grow, and it may appear that the monitoring process has somehow stopped. It is possible to check the program activity with the `touch` sub-option which causes the monitoring program to touch the output file at every step.

– **output file:**

The output file contains one data column for each waveform process variable named in the input file. The names of the data columns are given by the values of `WaveformName` in the input file. The units are obtained internally from the EPICS database. An additional long column `Index` is created that give the index of each point in the waveform.

The values of the scalar PVs are written to parameters with names given by the `ReadbackName` column of the optional scalars input file. The units are obtained internally from the EPICS database.

By default, the data type is float (single precision). Each reading step produces a new page in the output file.

Time and other miscellaneous parameters are defined:

* `Time` — Double column for elapsed time of readback since the start of epoch.
* `TimeOfDay` — Float column for system time in units of hours. The time does not wrap around at 24 hours.
* `DayOfMonth` — Float column for system time in units of days. The day does not wrap around at the month boundary.
* `Step` — Long column for step number.
* `CAerrors` — Long column for number of channel access errors at each reading step.

For each scalar PV defined in the `scalars` command line option a parameter of type double is defined.

Many additional parameters which don't change values throughout the file are defined:

  * `TimeStamp` — String column for time stamp for file.
  * `PageTimeStamp` — String column for time stamp for each page.
  * `StartTime` — Double column for start time from `C` time call cast to type double.
  * `YearStartTime` — Double column for start time of present year from `C` time call cast to type double.
  * `StartYear` — Short parameter for the year when the file was started.
  * `StartJulianDay` — Short parameter for the day when the file was started.
  * `StartMonth` — Short parameter for the month when the file was started.
  * `StartDayOfMonth` — Short parameter for the day of month when the file was started.
  * `StartHour` — Short parameter for the hour when the file was started.

- **switches:**
  - `-PVnames=<name>[,<name>]` — specifies a list of PV names to read. It the waveforms are of different lengths, the short ones are padded with zeros.
  - `-scalars=<filename>` — Specifies input file for scalar PV names. The values are logged as parameters.
  - `-erase` — If the output file already exists, then it will be overwritten by `sddswmonitor`.
  - `-generations[=digits=<integer>][,delimiter=<string>]` — The output is sent to the file `<SDDSoutputfile>-<N>`, where `<N>` is the smallest positive integer such that the file does not already exist. By default, four digits are used for formating `<N>`, so that the first generation number is 0001.
  - `-interval=<real-value>[,<time-units>]` — Specifies the interval between readings. The time interval is implemented with a call to usleep between calls to the control system. because the calls to the control system may take up a significant amount of time, the average effective time interval may be longer than specified.
  - `-steps=<integer-value>` — Number of readbacks for each process variable before normal exiting.
  - `-time=<real-value>[,<time-units>]` — Total time for monitoring. Valid time units are seconds, minutes, hours, and days. The program calculates the number of steps by dividing this time by the interval. The completion time may be longer, because the time interval in not garanteed.
  - `-verbose` — prints out a message when data is taken.
  - `-singleShot[=noprompt]` — a single read is prompted at the terminal and initiated by a `<cr>` key press. The time interval is disabled. With `noprompt` present, no prompt is written to the terminal, but a `<cr>` is still expected. Typing "q" or "Q" terminates the monitoring.
  - `-oncaerror=usezero|skiprow|exit` — Selects action taken when a channel access error ocurrs. The default is using zero (`usezero`) for the value of the process variable with the channel access error, and resuming execution. The second option (`skiprow`) is to simply throw away all the data for that read step, and resume execution. the third option is to exit the program.
  - `-ezcaTiming[=<timeout>,<retries>]` — Sets EZCA timeout and retry parameters.

- `-scalars=<filename>` — Specifies sddsmonitor input file to get names of scalar PVs from. These will be logged in the output file as parameters.
- `-conditions=<filename>,{allMustPass | oneMustPass}[,touchOutput][,retakeStep]]` — Names an SDDS file containing PVs to read and limits on each PV that must be satisfied for data to be taken and logged. The file is like the main input file, but has numerical columns LowerLimit and UpperLimit.

  One of `allMustPass` or `oneMustPass` must be specified. It would make sense to use `allMustPass` in most monitoring applications. If `touchOutput` is present, then the output file is touched, even if no data is written. This way, one can determine by the time stamp of the file whether the monitoring job is still alive when the conditions fail for a long period of time. If `retakeStep` is present, then the value of `Step` in the output file is not incremented until the conditions pass, and data is written to the output file.
- `-comment=<parameterName>,<text>` — Gives the parameter name for a comment to be placed in the SDDS output file, along with the text to be placed in the file.

- **see also:**
  - `sddsmonitor` (2.15)
  - `sddsvmonitor` (2.22)
  - `sddssnapshot` (2.18)
- **author: M. Borland and L. Emery, ANL**

2.24 squishPVs

- **description:** `squishPVs` minimizes the values of a set of readback process variables (for example bpm readbacks) by varying one setpoint process variable (for example a corrector magnet setpoint) which has a physical influence on the readback process variables. An input file defines one or more data sets of correction groups, each consisting of a list of readback PVs and one actuator PV. The method simulates the manual tweaking of physical devices that is often necessary when the readbacks are very noisy or when obtaining a reponse matrix is not worth the trouble.

  The name `squishPVs` comes from the apparent squishing of the real-time trajectory display when `squishPVs` is running.

- **example:** The first-turn horizontal trajectory of the APS ring is minimized with this command:

  ```
  squishPVs xTrajCorr.sdds
  ```

  where correction groups are defined in the file `xTrajCorr.sdds`. Part of the first correction group in the file `xTrajCorr.sdds` is shown below:

  ```
  SDDS1
  &description
   text = "Input file for first turn correction", &end
  &parameter
   name = "CorrectorPV",  type = "string", &end
  &parameter
   name = "Gain",  type = "double", &end
  ! LowerLimit and UpperLimit parameters are optional
  &parameter
   name = "LowerLimit", type = "double", &end
  &parameter
   name = "UpperLimit", type = "double", &end
  &column
   name = "BpmPV",  type = "string", &end
  ! Offset PVs are optional
  &column
   name = "OffsetPV", type = "string", &end
  ! Fixed offsets are optional and default to 0
  &column
   name = "OffsetValue", type = "double" &end
  ! Weights are optional and default to 1
  &column
   name = "Weight", type = "double", &end
  &data
   mode = "ascii", &end
  ! table number 1
  ! CorrectorPV:
  S1A:H1:CurrentAO
  1.000000000000000e+00                   ! Gain
  ```

```
-150                                  ! LowerLimit
150                                   ! UpperLimit
360                                   ! number of rows
S1A:P1:ms:x S1A:P1:ms:x:SetpointAO 0.1 1
S1A:P2:ms:x S1A:P2:ms:x:SetpointAO 0.2 2
S1A:P3:ms:x S1A:P3:ms:x:SetpointAO 0.3 1
S1A:P4:ms:x S1A:P4:ms:x:SetpointAO 0.4 1
S1B:P5:ms:x S1B:P5:ms:x:SetpointAO 0.5 2
S1B:P4:ms:x S1B:P4:ms:x:SetpointAO 0.4 1
...
```

- **synopsis:**

```
usage: squishPVs <inputfile>
    [-count=<pvName>,<lower>,<upper>,<number>]
    [-averages=<number>,<pauseInSeconds>] [-stepSize=<value>]
    [-subdivisions=<number>,<factor>]
    [-upstep=count=<number>,factor=<value>]
    [-repeat={number=<integer> | forever}[,pause=<seconds>]
    [-ezcaTiming=<timeout>,<retries>]
    [-settlingTime=<seconds>] [-verbose]
    [-criterion={mav | rms}] [-maximize]
    [-threshold=<value>] [-actionLevel=<value>]
    [-testValues=<file>,[limit=<number>]]
```

- **files:**
  - **input file:**
    The input file data pages define correction groups. A correction group consists of a list of readback PVs and one corrector PV. The required column and parameters are:
    * `BpmPV` — String column of readback PVs. The name `BpmPV` is due to the original application of `squishPV` where the first turn trajectory is reduced by tweaked by corrector magnets.
    * `OffsetPV` — Optional string column of PVs to subtract from corresponding `BpmPV`. This will have the effect of optimizing toward the values of the offsets for positive weights (the default). The default offsets are all 0.
    * `Weight` — Optional double-precision column of values by which to multiply the `readback-offset` values. The default weight is 1.
    * `CorrectorPV` — String parameter of the corrector PV used in the correction group.
    * `Gain` — Double parameter for factor by which to multiply the value of the command line option `stepsize`.

- **switches:**
  - `-count=<pvName>,<lower>,<upper>,<number>` — Optional. Specifies a condition after which a reading will be taken. The process variable `<pvName>` is read at 1 second intervals. If this PV is within the validity limits specified for this number of readbacks, then the PVs of the correction group are read.

- -averages=<number>,<pauseInSeconds — Optional. Number of PV readings to average and the number of seconds to wait between readings.
- -stepsize=<value> — Optional. Initial step size to attempt for corrector PVs.
- -subdivisions=<number>,<factor> — Optionally specifies number and size of interval subdivisions.
- -upstep=count=<number>,factor=<value> — Optionally specifies the number of steps to take in one direction before increasing the stepsize by the given factor.
- -repeat=number=<integer> | forever[,pause=<seconds>] — Specifies repeated optimization, either a given number of times or indefinitely, with a pause in between. This is useful if a system needs periodic tuning up.
- -settlingTime=<seconds> — Optionally specifies the settling time after corrector PV changes.
- -criterion=mav|rms — Optionally specifies mean-absolute-value or RMS reduction. Mean-absolute-value is the default.
- -threshold=<value> — Specifies that the change in the criterion must be below the specified value in order to be considered a genuine improvment.
- -actionLevel=<value> — Specifies that the criterion must be above the given value in order for optimization to start.
- -maximize — Specifies that the criterion should be maximized. The default is minimization.
- -ezcaTiming=<timeout>,<retries> — Optionally specifies EZCA timing parameters.
- -verbose — Optionally requests output during run.
- -testValues=<file>,[limit=<number>] – file is a sdds format file containing minimum and maximum values of PV's specifying a range outside of which the feedback is temporarily suspended. Column names are ControlName, MinimumValue, MaximumValue. Optional column names are SleepTime, ResetTime. limits is the maixum number of failure times. The program will be terminated when the continuous failure times reaches the limit.

• **see also:**
  - sddscontrollaw (2.8)

• **author: M. Borland, ANL**

References

[1] M. Borland, "Application Programmer's Guide for SDDS Version 1.4", APS LS Note.

[2] L. Emery, "Commissioning Software Tools at the Advanced Photon Source", to appear in *Proceedings of the 1995 Particle Accelerator Conference*, May 1995, Dallas.

[3] E. Norum, "Monopulse Beam Position Monitor Field Programmable Gate Array Function Description", unpublished. Available here.

Contents