The FILL_DB User's Guide

Steve Scott
March, 2005

Last updated April 12, 2006

New features (4/12/2006)

- Two new extraction keywords: 'stddev' and 'sdom'. These return the standard deviation of the variable over the defined measurement interval and the standard deviation of the mean.

FILL_DB is a utility that provide a convenient way to generate a table of plasma parameters from the C-Mod MDSPlus data system. It is patterned after the MINGL utilities developed about twenty years ago on TFTR by Dick Wieland.

To construct a database table using FILL_DB, you need to construct two files: one file contains a list of MDSPlus signals whose data are to be entered into the table, and and the other file contains a list of shots and times.

Table of Contents

Usage

To use the FILL_DB utility, type

```
IDL
.run /home/sscott/locus/fill_db.pro
fill_db
```

You will be prompted to enter:

- the name of the database;
- the name of the database table;
- the name of a shot-list file;
- the name of a parameter-list file;

These shot-list and parameter files are simple ascii text files that may be generated with your favorite text editor. The format of the shot-list file and parameter-list file are described below.

The table must be pre-defined for you by Josh. As described below, you can either ask him to create a table with 'generic' names or with user-specified names.

FILL_DB will then loop through the shots in your shot-list file, read the data from each waveform defined in your parameter-list file, and write the data into the table.

Execution time: FILL_DB takes a few seconds to process each line in your shotlist file, assuming that you have defined a reasonable number (~100) of waveforms in the parameter-list file.

Shot List File

The shot list file is a space-delimited ascii file that defines the list of shots to be processed and the time (and time window) at which data is to be read for each shot.

Each line in the shotlist file defines a shot number and a time. If you want to store data for multiple time points in a shot, each time point must appear on a separate line.

Typical lines in the shot list file:

```
1050403027      1.54          0.020
1050403028      t_disrupt     0.015   offset=-0.020
1050403029      t_emax        0.01    offset=-0.01   agas=4.  post_boron=6.
```

FILL_DB will extract data for shot 027 at time = 1.54 +/- 0.020 seconds. For shot 28, it will determine the disruption time, then extract data over a time period extending from 35 ms before the disruption to 5 ms before the disruption.

Beware! FILL_DB reads each waveform in succession, looking for data points in the time interval that you define. If it doesn't find any data points within your time interval, it enters NOTHING into the database. Currently, FILL_DB does NOT interpolate. So for example if you set the time = 1.01 with a time interval of 0.001, FILL_DB will only look for data from 1.009 to 1.011 seconds. With these timing parameters, it would not read any of the EFIT data, which is on a time axis 1.00, 1.02, 1.04 … seconds.

Update (11/17/2005): there is now a partial fix to this limitation. If your parameter extraction keyword is 'avg', i.e. if you are asking that the average value of the waveform be recorded into the database, you can specify that FILL_DB take special measures if the user-specified time window does not include any time points in the waveform:

- Interpolate: if you choose this option (it is a prompt when you run FILL_DB), then the waveform will be interpolated to the midpoint of the user-specified time window.

- Nearest: if you choose this option, FILL_DB will take the waveform value from the time point closest to the midpoint of the user-specified time window.

If you choose neither of these options, then FILL_DB will continue to put NOTHING into the record of waveforms which have no time points that

lie inside the user-specified time window. Also, please note that this option currently applies only to parameters that have the keyword = 'avg'.

T_disrupt is one of several 'event times' that are known to FILL_DB. The list of all event times supported by FILL_DB is presented below.

The entries agas=4. post_boron=6. define the values of the parameters agas and post_boron for this shot. Note that you can't define the value of an arbitrary waveform, i.e. you can't override a value that is retrieved through mdsplus. You can only define the value of a parameter through the shotlist if that parameter's tree name is defined to be 'user' in the parameter list. See the description under 'parameter list file' below.

Alternate entry of shot numbers: If you are content with defining a single time (or a single event-time) and a single averaging window that will apply to *all* shots, then you don't need to construct a shotlist file.

Instead, you define a *range of run days*, and FILL_DB will attempt to load data for all shots for all run days within the defined range.

To select this option, enter 'none' in response to FILL_DB's prompt for a shotlist filename. You will be prompted for the necessary information.

One additional feature of this option is that you can define one or more *selection thresholds* that exclude un-interesting shots from the database. For example, if you are building a database covering 100 run days, a significant number of the potential shots are uninteresting fizzles, and you might want to avoid cluttering your database table with such shots. You can define one or more waveform names and corresponding threshold values, e.g. you might choose to exclude shots which don't reach 0.5 MA by defining \magnetics:ip as the waveform name, 0.001 as the scale factor, and 0.5 as the threshold.

Also, you will be prompted for a list of name-value pairs. Use this opportunity to define the values for any user-parameters that you defined in the parameter list file (i.e. those which have the tree name = 'user'). For example, if you define two parameters 'agas' and 'wolfe_favorite' in the parameter list, you could define their values by typing

```
agas=4.   wolfe_favorite=3.
```

This will cause FILL_DB to define agas=4. and wolfe_favorite=3. for ALL of the shots that it processes in the list.

Note that if you want to have different values for these user-defined parameters on a shot-by-shot basis, then you need to prepare a real shot-list file, i.e. you can't use this alternate method of shot-number entry.

Parameter List File

This file defines the list of waveforms whose data is to be read and stored into the database table. Each waveform appears on a separate line.

Examples:

```
ip          cmod  \magnetics::ip                  avg      f6.3  scale=-1.e-6
prf         cmod  \rf::rf_power_net               max      f6.2  nodata=0.
max_wtot    cmod  \analysis::efit_aeqdsk:wplasm   maxmax   f6.2  scale=0.001
ddt_ip      cmod  \magnetics::ip                  ddt      f6.2  scale=-1.e-6
t_disr      cmod   t_disrupt                      avg      f6.2
agas        user   xxxx                          xxxx     f6.2  nodata=2.
```

ip — column name: this is the name by which the parameter will be known to LOCUS.

cmod — tree name: defines the MDSPLUS tree from which FILL_DB will read the data.

user — this is a special designation of the tree name. *It signifies that the user, rather than mdsplus, will supply a value for this parameter.*

The user will provide the value through the shotlist, in the format of name-value pairs. For example, in the shotlist file, the user could append the line for shot 105061000 with the string: 'agas=4.' to indicate that helium was used for that shot.

Note that if the tree name = 'user', then the name of waveform and the extraction keyword are superfluous. So you should just put some nonsense entry such as 'xxxx' for the waveform name and extraction keyword, as shown in the example above.

\magnetics::ip — defines the full path name of the waveform

t_disrupt — FILL_DB also defines a number of specific 'event times' whose value can be written into the database table. The complete list of event times is presented below.

avg — `extraction keyword': this defines how FILL_DB is to extract a value from the time window that is defined in the shot list file. Typically the user wants the average value, but a number of other options are available, including time derivatives etc. The complete list of extraction keywords is presented below.

f6.3 At present, this field is mandatory but is ignored by FILL_DB. Eventually, it will be used to (a) define whether the parameter is an integer or a real number; and (b) the format for printing out the parameter when the user asks for tabular rather than graphical output in LOCUS.

Scale=-1.e-6 *scale* is an optional keyword that defines a multiplicative factor to be applied to the data before storage into the database table.

nodata=20. *nodata* is an optional keyword that defines the value that will be stored for that parameter in the event that FILL_DB has problems reading the waveform.

By default, if FILL_DB has some difficulty reading a parameter, it stores a very big number (1.5e37) into the database table for that parameter. LOCUS basically ignores all numbers greater than 1.e37. For example, if FILL_DB couldn't find a waveform for the RF power for a given shot, it would store the RF power = 1.5e37, and then records which have RF power greater than 1.e37 would be ignored by LOCUS. Specifically,

- If you ask to *plot* the RF power, shots with RF power greater than 1.e37 would not be plotted.
- If you use the RF power as a *constraint*, then any shot with RF power in excess of 1.e37 would be *excluded* from the search.

This latter behavior is not necessarily appropriate for all situations. For example, in LOCUS suppose you want to look only at Ohmic shots. If your parameter list stored the RF power under the name PRF, you might define "PRF<0.01" as a constraint. But possibly some shots didn't have any RF power and didn't have any RF waveforms either. These shots would get excluded from your search.

There are two ways to deal with this problem. You could define a constraint "(PRF<0.01)or(PRF>1.e37)" – this would work, but is a little klugy. Instead, in your parameter list you could define NODATA=0. in which case shots without RF waveforms will have 0. stored as the RF power. Then your simple constraint "PRF<0.01" will work fine.

Special Waveform Names (added May 2005)

The following waveforms are derived from a dwscope:

/home/Marmar/scopes/g_confinement_plasma_w_dwdt_tauiter89.dat

Note that the confinement time includes dW/dt effects, and I each of the following waveforms assumes that 80% of the RF power is absorbed.

zz_taue	energy confinement time, sec
zz_taue89	tauE predicted from the Lmode89 regression
zz_taue98	tauE predicted from the Hmode98 regression
zz_taue89_ratio	ratio of tauE to the taue89 regression
zz_taue98_ratio	ratio of tauE to the taue98 regression
zz_poh	Ohmic heating power
zz_ptotal	total heating power
zz_pthreshold_ratio	ratio of heating power to the H-mode threshold power
zz_betan	normalized beta, from /home/granetz/scopes/scope_efit_new.dat
hmode	=1 if algorithm thinks the plasma is in H-mode, and 0 otherwise. The algorithm is based on the ratio of two edge SXR channels; if the ratio exceeds the value set by the optional keyword *threshold*, then the plasma is defined to be in H-mode. If the optional keyword 'threshold' is not set by the user, then a threshold value of 1.00 is used by the algorithm. Note: as of 6/15/2005, the algorithm to identify periods of H-mode is good but not great.

For the following four special waveforms, please consider using the optional keywords *threshold* and *min_hmode_duration*. The algorithm will ignore any H-mode it locates whose duration does not exceed min_hmode_duration; if this parameter is not set, then the algorithm imposes a minimum of 40 ms.

hmode_1_duration	duration, in seconds, of first hmode.
hmode_2_duration	duration, in seconds, of second hmode
hmode_3_duration	duration, in seconds, of third hmode
hmode_4_duration	duration, in seconds, of 4th hmode

Other special waveform names

shot_day	this will store the value of shot – 1000*(shot-1000), which effectively stores the shot number relative to the beginning of the day, e.g. it would store a value of 6 for 1050704006
shot_counter	

External waveforms

Note: as of July 8, 2005, this capability is untested.

Despite Martin Greenwald's finest efforts, not all C-Mod data is stored in the MDS-Plus tree. Fill_DB provides a capability to interface with an IDL procedure that YOU write to fetch data into the database table.

The main limitation of this capability is that your home-grown IDL procedure must strictly follow the list of input/output parameters described below.

pro my_procedure, ishot, tstart, tend, value_type, ext_value_1, ext_value_2, value, status

inputs

ishot	shot number (long integer)
tstart	starting time of the averaging interval (sec)
tend	ending time of the averaging interval (sec)
value_type	this is a character string, which you will probably just ignore. It has values such as 'avg', 'min', 'max', 'ddt', etc which for standard signals allow the user to ask that the code return the average value of the signal over the [tstart,tend] time window, or the minimum value, or the maximum value, etc. You may or may not want to provide similar support via your procedure.
ext_value_1	this is an arbitrary input parameter (float) is specified by the optional parameter 'ext_value_1' in the parameter table. See below for an example. You might use this to define a channel number, for example.
ext_value_2	another arbitrary input parameter.

outputs

value	the value of whatever it is you want stored in the database table
status	0 for success, 1 for any problem (status is an integer).

Your procedure *must* include all of these parameters, even if you don't use them internally. So you may either need to rewrite your existing procedure so that it has these parameters, or, more likely, write an interface procedure that has the list of parameters that I want to see, and then internally it can make a call to your existing procedure with the parameter list that it wants to see.

In your 'parameter list' file, you would then have lines like

```
prad        cmod       \spectroscopy::twopi_foil              avg     f6.2   scale=1.e-6
prad_3      cmod       \spectroscopy::prad_2pi                avg     f6.2
my_val_1    external    my_procedure_name_1                    avg     f6.2    ext_value_1=35.
```

```
my_val_2   external   my_procedure_name_2              avg     f6.2    ext_value_1=35.  ext_value_2=4.73
my_val_3   external   my_procedure_name_3              avg     f6.2    scale=1.e-3
```

where the first two lines are standard reads from the cmod tree, and the last three lines use the tree name ‘external’ to tell fill_db that it should make calls to the procedures ‘my_procedure_name_1’ or ‘my_procedure_name_2’ to fetch the data.

- My_procedure_name_1.pro actually makes use of the input parameter ext_value_1, so you have provided a value for it in the parameter table.

- My_procedure_name_2.pro uses both optional parameters, so you need to provide two values in the parameter table.

- My_procedure_name_3.pro doesn’t use any of the optional parameters.

Note the use of the optional scale factor in the last line … after you fill_db fetches a value from say my_procedure_name_1, I would multiply it by 1000. before putting it into the database table.

Existing external waveforms

Name	Returns	ext_value_1	ext_value_2
/home/sscott/locus/get_br.pro	radial field	rmajor (m)	z (m)
/home/sscott/locus/get_bz.pro	vertical field	rmajor (m)	z (m)
/home/sscott/locus/get_bt.pro	toroidal field	rmajor (m)	z (m)

These routines are simply shells around /home/wolfe/vms-idl/idl/cmod/big_fields.pro. They are designed to return the radial and vertical fields well outside the plasma. If you specify a location close to the plasma, e.g. rmajor=1.1, z=0., you will get lots of warning messages reminding you that you should use a different routine.

If rmajor < 1.15 meters, the procedure for toroidal field (get_bt.pro) simply returns 0.66*(\magnetics::btor)/rmajor, i.e. if the user-specified position lies inside the major radius of the outer leg of the TF coil. If rmajor is greater than 1.15 m, it applies a crude TF ripple model. This model assumes that the location is exactly mid-way between two TF coils.

Extraction keywords

avg	average value during the time window
max	maximum value during time window
min	minimum value during time window
first	value at start of time window
last	value at end of time window
after	average value over a time period 5-15 ms after end of time window
before	average value over a time period 5-15 ms before start of time window
max_delta	difference between maximum value and minimum value during time window
max_ratio	ratio of maximum value to minimum value during time window
delta	difference between value at end and start of time window
ratio	ratio of parameter value at end of time window to value at start of time window
ddt	we define the time derivative entirely by the values at the start and end of the time window, i.e. ddt = (ylast-ystart)/(time_last-time-time_start)
deriv	use the built-in IDL procedure "derive" to compute the time derivative. Note that you may want to smooth the data with the keyword "dt_smooth" in conjunction with this routine.
scalar	This extraction keyword informs FILL_DB that the waveform in question is a scalar, i.e. it has no time dependence.
stddev	standard deviation of the data during the time window. If there is only a single data point in the time window, the value of zero is returned.
sdom	standard deviation of the mean of the data during the time window, which is just the standard deviation divided by sqrt(nn-1) where nn is the number of measurements. If there is only a single measurement in the time window, a value of zero is returned.

The following extraction keywords will cause FILL_DB to search the waveform throughout the entire shot rather than just at the time window specified in the shot list.

time_at_max	time at which parameter reaches a maximum *throughout the entire shot*
minmin	minimum value *throughout the entire shot*
maxmax	maximum value *throughout the entire shot*
maxmax_ddt	maximum value of time derivative *throughout entire shot*
minmin_ddt	minimum value of time derivative *throughout entire shot*
max_sustained	maximum value that is sustained continuously for a defined period. The period is defined by the optional keyword 'threshold'. For example, if threshold = 0.050, then this would return e.g. the maximum RF power that is sustained for a period of at least 50 msec. Warning: using this keyword is considerably more cpu-intensive than other keywords.

t_start earliest time *throughout the entire shot* at which the waveform exceeds a threshold value. This threshold value is specified with the keyword "threshold", e.g. For example, the following line in the parameter list file

tstart_ip cmod \magnetics::ip t_start f6.3 scale=-1.e-6 threshold = 0.02

will store the time at which the plasma current first exceeds 0.02 MA into the column name 'tstart_ip'. Note that the order of the two keywords "scale" and "threshold" at the end of the line can be in any order.

Note: 't_start' and 'tend' allow the user to define on and off times for any waveform. But this new functionality is limited to *storing values of the corresponding on/off times* – it does not provide the capability to define new 'event times' at which FILL_DB can be asked to extract other waveforms.

For example, you could define a column 'tstart_nl4' with reference to the waveform for TCI channel 4 with threshold value, say, 1.2e20. This would populate a column that stores the time at which the two-color interferometer reports the NeL first rises above 1.2e20. But at present there is no way you can ask FILL_DB to extract data for all of the other waveforms at this time.

t_end latest time throughout the entire shot at which the waveform exceeds a threshold value. This threshold value is specified with the keyword "threshold". See t_start above for details.

integrate_from_t0 integrates the value of the waveform over time, from t=0 to the time of interest as specified in the shotlist file. For example, this might be useful if want the total number of joules injected by the RF system at the time of interest.

integrate_from_start integrates the value of the waveform over time, starting from whenever the waveform starts (which may or may not be at t=0.) to the time of interest as specified in the shotlist file. For example, this might be useful if you wanted to know the total amount of gas injected prior to the time of interest.

rfstart_ddt This allows you to take time derivatives starting at the time that RF heating starts. The best way to illustrate its use is with an example. The following entry in the parameter file would create a variable rfddt_200_prad, which stores the time derivative of the radiated power in the first 200 ms of RF heating.

rfddt_200_prad cmod \spectroscopy::twopi_foil rfstart_ddt f6.2 scale=1.e-6 event_time_or=t_rfon time_offset_or=0.2

Note that, although the keyword is called “rfstart_ddt”, you actually define the event using the event_time_or=t_rfon. So for example, if you had selected event_time=t_rfoff in the example above, then Fill_db would fill the variable with the time derivative of the radiated power in the first 200 ms after the *end* of RF heating.

The actual computation of the time derivative is the same as the “ddt” extraction keyword.

rstart_deriv works exactly the same as rfstart_ddt, except that the algorithm to compute the derivative is taken from the “derive” keyword.

Event times

`Event times' have two distinct purposes.

First, in the shot-list file, an event time (e.g.t_disrupt) can be substituted in place of a fixed time (e.g. 1.54 sec). If an event-time is defined for a given shot in the shot list file, then FILL_DB will retrieve and store plasma parameters at the time at which the chosen event occurs. If the event does not occur, for example if the user chooses `t_disrupt' in the shot list file but there is no disruption in a given shot, then no data is stored for that shot.

1040509026 t_disrupt 0.02 offset=-0.030

In this example, FILL_DB would find the disruption time for shot 026, and then extract and store plasma data from a time window 50-10 ms before the disruption, i.e. at a time 30 ms before the disruption with an averaging window +/-20 ms.

A lot of the power and convenience of FILL_DB derives from its ability to extract and store data at these 'event times'. The user is spared the drudgery of looking over shots manually to identify appropriate times to extract the data.

Second, in the parameter file, an event time can be substituted for a waveform name. For example, the line below would cause FILL_DB to fetch the disruption time for each shot and store it under the column name 't_disr'.

t_disr cmod t_disrupt avg f6.3

Note that when an event-time is defined in this way, FILL_DB ignores the extraction keyword 'avg' – it would make no difference if the extraction keyword had been 'max' or 'min'.

The following event times are currently supported

t_ipflat_start start of `flattop' on Ip. This is the earliest time that the plasma current reaches 97% of its maximum value + 20 ms. To be considered a flattop, the duration over which Ip remains within 3% of its maximum value must exceed 200 ms.

t_ipflat_end end of flattop on Ip: latest time at which Ip remains above 97% of its maximum value, minus 20 ms.

t_btflat_start start of Bt flattop: earliest time at which Bt remains above 98% of its maximum value, plus 50ms.

t_btflat_end end of Bt flattop: latest time at which Bt remains above 98% of its maximum value, minus 70 ms.

t_disrupt earliest time at which d(ip)/dt exceeds 100 MA /second

t_emax time at which energy content in plasma is maximum (as measured by \analysis::efit_aeqdsk:wplasm'

t_rfon earliest time at which RF power exceeds 0.05 MW
t_rfoff latest time at which RF power exceeds 0.05 MW
t_ohmic if there is some auxiliary power in the plasma, this time will be defined as occurring shortly (30ms) before the start of the auxiliary power. If there is no auxiliary power, it will be defined as being shortly before the end of the earlier of the I_p flattop or the B_T flattop. We'll have to deal with the case of no flattops separately.

t_dnbon starting time of DNB
t_dnboff ending time of DNB

t_wrf time at which the total number of megajoules injected by the RF system exceeds a threshold value. The threshold value *must* be defined by the optional keyword wrf_threshold. If you ask for the event time t_wrf but fail to define the threshold value wrf_threshold, FILL_DB will issue an error and will stop.

Note: the following three event-times are untested as of 6/15/2005. There is a good chance that they will have some problems with spurious data, i.e. the various values of beta may reach a maximum at very early or late in the discharge when the magnetic field is very small.

t_betat_max time of maximum toroidal beta
t_betap_max time of maximum poloidal beta
t_betan_max time of maximum normalized toroidal beta

Note: for the following event times related to the start and end of the H-modes, you should define the optional keyword *min_hmode_duration*. If you set min_hmode_duration=0.060 for example, the code will ignore all H-modes which do not persist for at least 60 ms. If you do not set this parameter, the code defaults to a minimum H-mode duration of 40 ms.

Also, you might want to define the value of optional keyword *threshold* when working with the event times that are tied to the H-mode timing. In the algorithm that looks for H-modes, *threshold* controls the ratio of the two edge soft x-ray channels that must be exceeded for the algorithm to regard the plasma as being in an H-mode state. If you do not specify a value for threshold, the code uses a value of unity.

t_hmode1_start start time of the first H-mode
t_hmode2_start start time of the second H-mode
t_hmode3_start start time of the third H-mode
t_hmode4_start start time of the fourth H-mode

t_hmode1_end end time of the first H-mode
t_hmode2_end end time of the second H-mode
t_hmode3_end end time of the third H-mode
t_hmode4_end end time of the fourth H-mode

The following event times are also tied to the Hmode times, but in this case, the start/end times of the Hmodes are not determined by an algorithm. Instead, a user has manually created an ascii file *user_hmode_times.txt* that lists the Hmode times for each shot.

t_hmode1_manual_start start time of the first H-mode
t_hmode2_manual_start start time of the second H-mode
t_hmode3_manual_start start time of the third H-mode
t_hmode4_manual_start start time of the fourth H-mode

t_hmode1_manual_end end time of the first H-mode
t_hmode2_manual_end end time of the second H-mode
t_hmode3_manual_end end time of the third H-mode
t_hmode4_manual_end end time of the fourth H-mode

As of August 23, this capability has not been implemented.

The following event times may be supported in the future

t_lhon starting time of LH power

t_lhoff ending time of LH power

t_nmax time of maximum neutron emission

t_pauxon minimum of t_rfon and t_lhon

t_pauxoff maximum of t_rfoff and t_lhoff

Fields in the Database Table

By default, FILL_DB always tries to write the following field names into the database table:

shot	shot number (IDL long)
dbkey	unique integer that identifies each record
times	time in seconds (float)
timems	time in milliseconds (IDL integer)
timeus	time in microseconds (IDL long)

The integer fields *timems* and *timeus* are provided as a convenient means to constrain data searches.

The remainder of the fields that FILL_DB tries to write will be defined in one of two ways.

In the 'generic' mode of operation, FILL_DB stores all of the variables under the *generic* names xx0, xx1, xx2, …

FILL_DB And LOCUS	actual database table
shot	shot
dbkey	dbkey
times	times
timems	timems
timeus	timeus
ip	xx0
bt	xx1
t_disr	xx2
wtot	xx3
…	

This mode of operation is appropriate if you intend to use the IDL procedure LOCUS to plot the data. LOCUS will prompt you for the name of the parameter list that created the table, so it will know the association between the convenient names (ip, bt, …) and the generic names (xx0, xx1, …)

In LOCUS, you will always refer to the parameters only by their convenient names, and the fact that the parameters are stored under a different set of names in the actual database table will be invisible to you.

The advantage of this mode of operation is that you might ask Josh to create, say, five generic database tables. Each table will have the *same* column names, so Josh's work is minimized. But then you can control what actually goes into the table with FILL_DB. You could populate the table with a parameter list

params_001.txt, but then erase the table and re-populate it with a different parameter list params_002.txt tomorrow. From your perspective you have changed the list of columns in the table, but from Josh's perspective you haven't – the column names actually stored in the database remain xx0, xx1, … so you don't have to ask Josh to create a new table each time you want to add or remove a column from your table.

The disadvantage of this mode of operation is that the column names in the table really are xx0, xx1, … so unless you the IDL procedure LOCUS to plot the data, the column names aren't going to be very informative.

`Personal' field names

Note: the capability to generate database tables using 'personal' field names, as described below, was removed on November 18, 2005, due to lack of use. If you want this capability restored, please contact Steve Scott.

If your intention is to use a utility other than LOCUS to process or plot your database table, you should instead store the columns under your 'personal' names as described below

In the *personal* mode of operation, FILL_DB will attempt to write columns exactly as you personally define them in the parameter list file. In the example above, the columns would be

shot
dbkey
times
timems
timeus
ip
bt
t_disr
wtot

If you select this option, then the actual column names used by the database table would be 'shot, ip, …" and so the names would be meaningful even if you were to use a utility other than LOCUS to plot the data.

The disadvantage of this mode is that Josh needs to define the column names *exactly* as you intend to use them when he creates the table. If you want to add a new column name to the table, you need to ask Josh to alter the table definition.

Optional Keywords

scale — multiply the signal by this factor

nodata — store this value if data is missing

dt_smooth — smooth data over this time period prior to storing or processing it.

Note: this capability was added to improve the performance of the Maximum_sustained extraction keyword for the RF power. There is considerable noise on the RF signal, so it is useful to smooth it by a millisecond or two prior to computing the maximum sustained value.

threshold — used by several extraction keywords, e.g t_start, max_sustained.

xdevice — defines the signal to be used for the time axis, for those cases where the time axis can't be obtained through the usual dim_of( ).

back_average — Extends the averaging interval backwards in time. For example, suppose the event time is t_emax = 1.22 sec, the offset is -0.02 sec, and the averaging interval is 0.01 sec. The actual averaging time would then be 1.19 – 1.21 sec.

But if the user also defined back_average = 0.030, then the averaging time would be 1.16 – 1.21 sec.

This is most useful if you want to record variables which might have an extended effect on the instantaneous plasma performance, such as the RF power or the radiated power.

min_hmode_duration — This optional keyword is applicable only when you ask for an event time that is tied to the timing of H-modes in the shot, e.g. t_hmode2_end. This keyword defines the minimum duration of an H-mode. If you set this parameter to say 0.080, then the algorithm will ignore all H-modes whose duration is less than 80 milliseconds.

The following four optional keywords can be used to *override* the time at which data is extracted from waveforms. Usually, the time at which data is extracted is controlled through the shot list file, or (if the same time-of-interest is desired for a long series of shots) through a direct prompt by Fill_DB. For example, one might construct a database table containing data at 1.0 seconds, or at the time of peak stored energy.

But suppose you want to correlate plasma behavior at different times in the discharge. For example, suppose you want to correlate the energy confinement time at time of peak stored energy with the impurity content in the Ohmic phase of the discharge. To

do this, you need to be able to allow each record in the database table to contain data from different times in the discharge. This can be done by using the following time-override optional keywords in the parameter file:

event_time_or	re-defines the time at which data is extracted to be a given event_time. For example, event_time_or=t_ohmic would cause data to be extracted at the Ohmic time.
dt_or	re-defines the time window over which data is extracted. Remember that the window is extended both forward and backward by this duration, i.e. dt_or=0.030 would create a 60 ms time window, not a 30ms time window.
time_offset_or	re-defines the offset time from the event_time_or. For example, choosing event_time_or=t_rf_on and time_offset_or=0.2 would cause the new time-of-interest to be 200 ms after the start of RF heating.
time_or	re-defines the time at which data is extracted. For example, time_or=1.2 seconds would cause the data to be extracted at 1.2 seconds for this variable.

Note: you can define a value for *either* event_time_or *or* time_or, but not both. It wouldn't make sense to ask that data be extracted at say 1.2 seconds and simulataneously ask that it be extracted at the time of peak stored energy.

Examples

The following session with FILL_DB shows its usage to record data into a table named *sds4* using a list of parameters from */home/sscott/locus/parameters_10.txt* from all shots taken between July 1, 2005 and July 18, 2005. Only one time point is recorded per shot, covering a period 20ms before the time of peak stored energy to 0ms before the time of peak stored energy.

Note in this example that no shot-list file is used. The user asks that all shots from a given range of rundays be included in the table.

Near the end of this session, the user asks that one condition be checked before data from a given time point is inserted into the database: the plasma current must be greater than 0.5 MA. This basically prevents fizzles from being recorded into the table.

This session was started from /home/sscott/locus. User input is highlighted in blue.

```
IDL
IDL>  .run fill_db.pro
IDL>  fill_db
Enter database name:                          [logbook]:    <cr>
Enter table name:                                [sds1]:    sds4
Enter initial value of shot counter           [       0]:    <cr>
Enter 1 to print detailed output:             [       0]:    <cr>
Enter name of shotlist file (or “none”):   [shotlist_001.txt]:  none

    OK, you may now specify a range of rundays and a range of shots for each runday

    Enter the starting rundate:                                [1020924]:  1050701
    Enter the ending rundate:                                  [1050701]:  1050718
    Shall I wait for incoming shots (1=yes)                    [       0]:  <cr>
    Shall I look for multiple times throughout the shot (1=yes)[       0]:  <cr>
    Shall I look for multiple times near end of shot (1=yes)   [       0]:  <cr>
    Shall I look for multiple times near start of shot (1=yes) [       0]:  <cr>
    List of space-delimited variable=value                     [   blank]:  <cr>
    Minimum shot number to examine on each day:                [       1]:  <cr>
    Maximum shot number to examine on each day:                [      50]:  <cr>

    Names of available event-times:

      t_ipflat_start    t_ipflat_end    t_btflat_start    t_btflat_end      t_disrupt
      t_emax            t_betap_max     t_betat_max       t_betan_max         t_rfon
      t_rfoff           t_dnbon         t_dnboff          t_ipstart          t_ipend
      t_jog1            t_ohmic         t_hmode1_start    t_hmode1_end       …
```

```
    Enter an event time name or a fixed time (e.g. 1.57)        [t_emax]:    t_emax
    Enter an offset time relative to this event                [-0.01000]:    -0.010
    Enter delta-t for time window (+/-)                         [0.01000]:   0.01

Enter name of parameter file file:                    [parameters_1.txt]:   parameters_10.txt
Enter filename of log file:                            [db_populate.log]:  <cr>
Enter 1 to store data under personal field names                [ 0]:  <cr>

You may now define minimum conditions for a shot to be entered into the database table

    Select a condition (-1 to terminate entry)                 [ -1]: 1
    Enter minimum threshold value:                             [ 0.300]:  0.5

    Select a condition (-1 to terminate entry)                 [ -1]  <cr>
```

Note: the desired time window was the 20ms interval preceding the time of peak stored energy. To get this, the user asked for the event-time = t_emax, then asked for a time offset of -10 ms relative to this event, then asked for a time window of 10ms on either side of the shifted time

Analyzed MSE data

Analyzed MSE data needs special treatment for two reasons. First, a total of nine different analyses (corresponding to different choices of analysis parameters) can be stored for each shot. So the user must specify which analysis 'try' should be read into the database table. Second, MSE data is available for ten spatial channels, so the user must also specify which channel is to be read.

Example: reading MSE time-dependent analysis data

a44_sub_ch10 mse beam:A44_sub avg f6.3 try=4 channel=9

Note that the tree=mse is what tells FILL_DB to treat this as analyzed MSE data. The optional keywords 'try' and 'channel' then specify the analysis try number and the channel number. The actual waveform that would be read in this example is \dnb::top.mse_pppl.analysis2.fft4.beam:a44_sub.

Example: reading MSE scalars (no time dependence)

A40_avg mse beam:a40_sub_avg scalar f6.3 try=2 channel=7

Here, the 'scalar' extraction keyword directs FILL_DB to read the waveform as a set of scalars and to extract the data from the appropriate channel.

EFIT pitch angles (New capability added March 2006))

You can read in the magnetic field pitch angle as computed by EFIT for the plasma radius corresponding to a given MSE channel location. The tree name is 'efit' and the allowable device names are listed below.

The radii corresponding to a given MSE channel are hard-wired into the source code of FILL_DB.PRO.

Examples:

efit_pitch_1	efit	efit_pitch	avg	f6.3	channel=1
efit_ratio_2	efit	efit_ratio	avg	f6.3	channel=2
efit_bz_4	efit	efit_bz	avg	f6.3	channel=4
efit_bt_2	efit	efit_bt	avg	f6.3	channel=2
efit_br_7	efit	efit_br	avg	f6.3	channel=7

Here, 'efit_pitch' is the pitch angle in degrees and 'efit_ratio' is bz/bt

A sample parameter file that includes some EFIT pitch-angle variables is /home/sscott/locus/mse_efit.parameters.

MFLUX pitch angles (New capability added March 2006))

You can read in the magnetic field pitch angle as computed by MFLUX for the plasma radius corresponding to a given MSE channel location. The tree name is 'mflux' and the allowable device names are listed below.

The radii corresponding to a given MSE channel are hard-wired into the source code of FILL_DB.PRO.

Important note: unlike the EFIT capability, the current support for MFLUX data allows only a single time-slice to be processed. That is, the MFLUX data which is stored in the DNB tree was computed for a single time point only. ***FILL_DB returns the MFLUX data from this time slice, irrespective of what time the user defined in the shotlist file.*** So it is the user's responsibility to make sure that the time of the MFLUX analysis is the same as the time of other parameters in the database table. Also note that MFLUX must be run manually – it is not generated automatically on every beam-into-gas shot.

For data taken before March 21, 2006, this typically isn't a problem, since Howard typically ran MFLUX at the time that the DNB fired. Once the DNB starts taking long pulses, we will have to upgrade this system to extract data from time-series MFLUX data.

Examples:

mflux_pitch_1	mflux	mflux_pitch	avg	f6.3	channel=1
mflux_pitchtan_6	mflux	mflux_pitchtan	avg f6.3 cahnnel=6		
efit_ratio_2	mflux	mflux_ratio	avg	f6.3	channel=2
mflux_bz_4	mflux	mflux_bz	avg	f6.3	channel=4
mflux_bt_2	mflux	mflux_bt	avg	f6.3	channel=2
mflux_br_7	efit	mflux_br	avg	f6.3	channel=7

Here, ‘mflux_pitch’ is the pitch angle in degrees and ‘mflux_pitch_map’ is the pitch angle mapped into the MSE coordinate frame of reference. I forget what mflux_pitchtan represents.

A sample parameter file that includes some MFLUX pitch-angle variables is /home/sscott/locus/mse_efit.parameters.

Remote mdsserver (new feature added October 14, 2005)

The first question that FILL_DB now asks you is whether you want to connect to a remote mdsserver. If you are working at the PSFC and if you want to access Alcator data, the answer is NO (which you indicate by entering a 0 or a <cr>).

Enter a '1' in response to this prompt if you want to connect to a remote mdsserver, i.e. if you want to access data from GA, PPPL, JET, etc.

You will then be prompted for the name of the remote mdsserver. The following is a list of mdsserver names:

atlas.gat.com
alcdata.psfc.mit.edu

Again, if you already at the PSFC, you do NOT need to ask for a connection to a remote server. You would need to do this only if you were at a computer at say PPPL and you wanted to access Alcator data.