Ed. note: This document assumes that the user has a working copy of SDT available. For some guidance regarding the installation and use of SDT, see the FAST website: https://sprg.ssl.berkeley.edu/fast/scienceops/fast_sdt_help.html and http://sprg.ssl.berkeley.edu/fast/scienceops/docs/fast_sdt_help.pdf. As of 2026-03-10, my own SDT installation (on Ubuntu) fails; it may need to be recompiled from source code.--jm

SDT_NOTES AND DESCRIPTIONS OF FAST EFI DATA FILE GENERATION.

This document is intended as a quick User's Guide for some basic SDT instructions and includes a description of the SDT/IDL software that can be used to create CDF files. Also, right at the end, included are some instructions on how we can change CDF attributes in files without reprocessing.

SETTING UP SDT:

At UCB-SSL, SDT has institutional support, and there is a working copy of SDT available for the local Linux workstations. The first step is to run a setup script. A sample script is included with the SDT software. This needs to be edited for your local system; note that the script includes both SDT and IDL setup.

The IDL setup should be commented out, and users should use their local IDL SPEDAS setup. (See: https://themis.ssl.berkeley.edu/software.shtml for SPEDAS setup instructions.) An example of a stripped setup for our local SSL environment can be found at https://sprg.ssl.berkeley.edu/~jimm/sdt_stuff/setup_sdt.

In this document, everything is done using csh scripts, but a Bourne shell setup is available with the SDT download.

For the C-shell, start with:

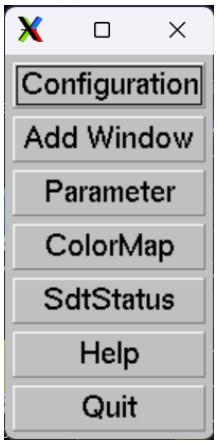
source setup_sdt

Then just call sdt

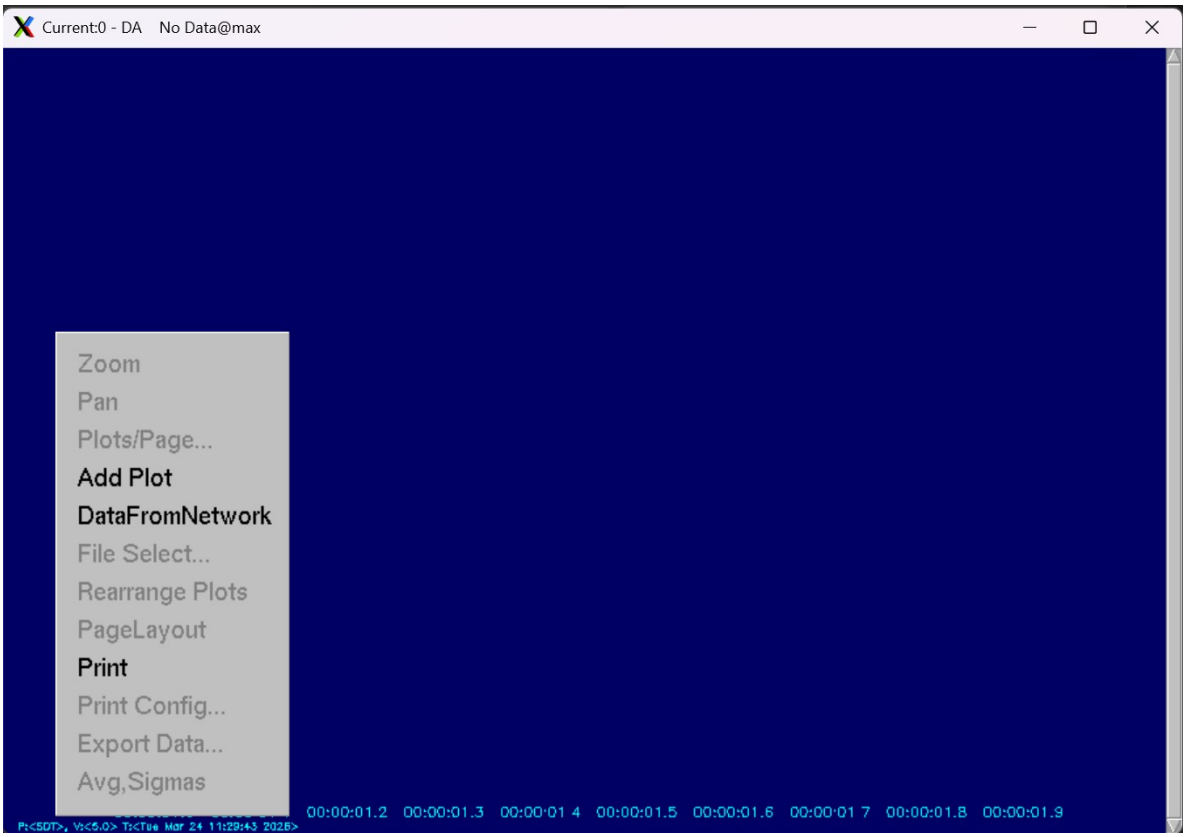
sdt &

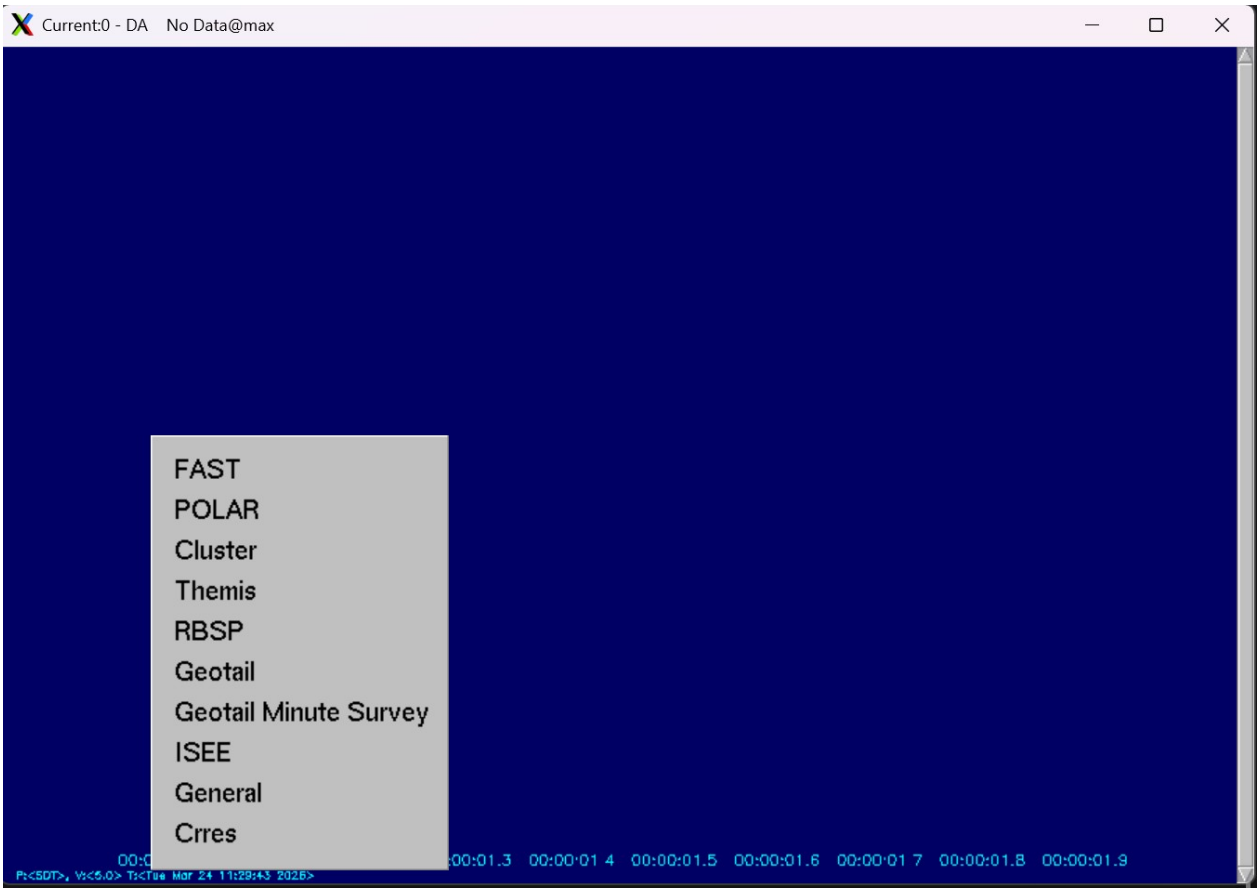
PLOTTING DATA:

That was easy. A menu pops up.

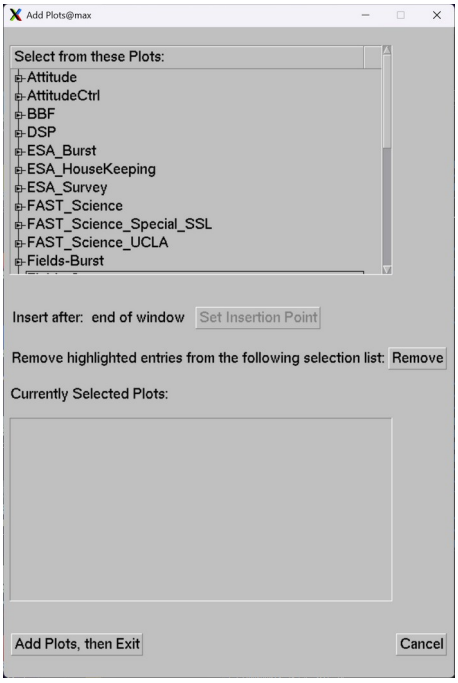


This menu will be up in the corner for the full SDT session. Mostly SDT will use 'Configuration' files, but we'll start here from scratch, and create a configuration. So first click on 'Add Window', and get a blank window. Right-click on the window, and another menu pops up, mostly with unavailable options. Right-click on 'Add Plot', and see another pop-up menu, which gives options for the available data sets.



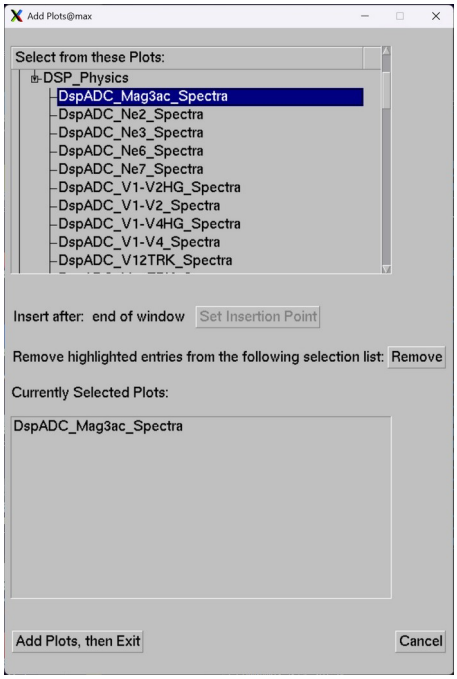


We want to work with FAST data first, so click on 'FAST', and get the next menu. This shows the available data types for the FAST spacecraft.

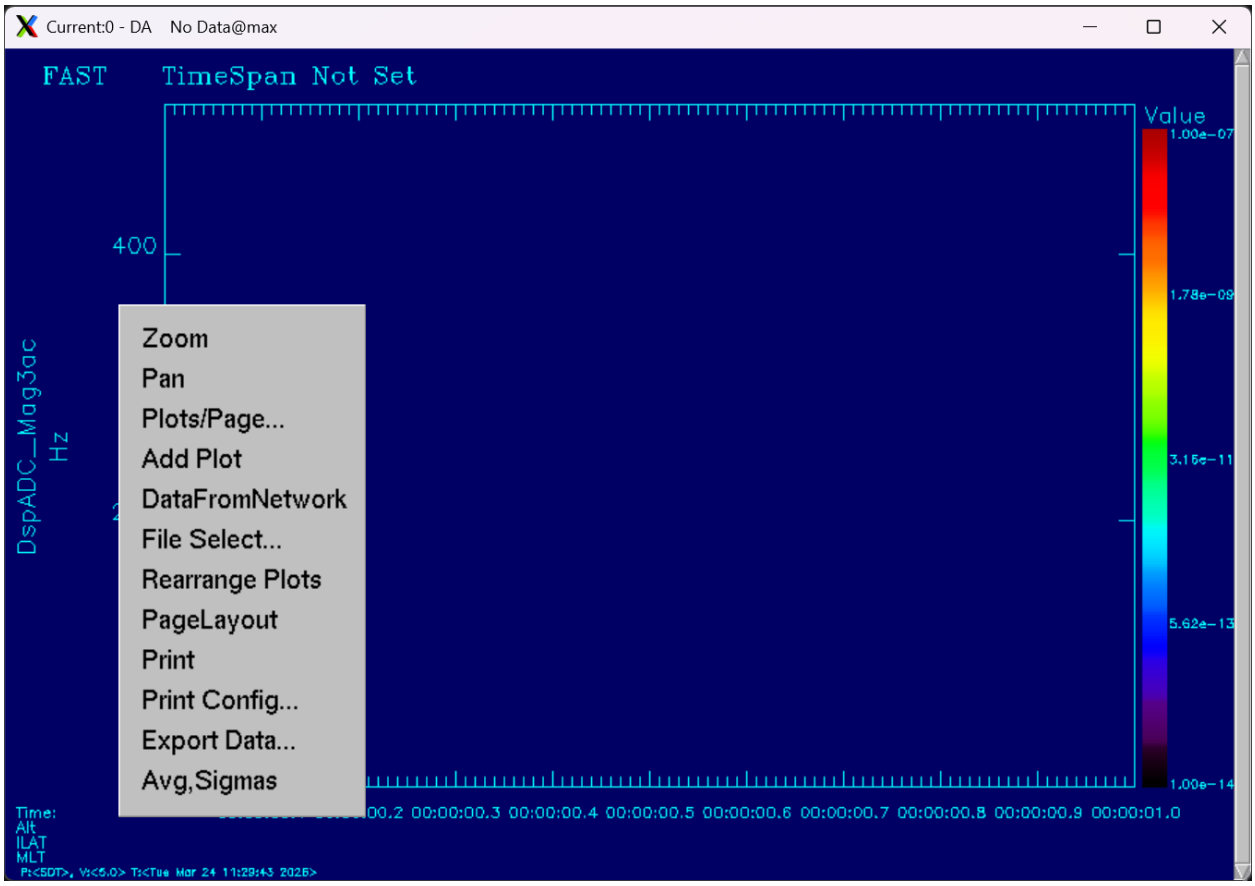


There is no metadata available from the SDT program, but you can get a list of each kind of data ('DQD', for 'Data Quantity Designation?') online: https://sprg.ssl.berkeley.edu/~jimm/sdt_stuff/FastDQD.txt. It is a good idea to become familiar with instrument terminology by accessing the FAST website: https://sprg.ssl.berkeley.edu/fast. Most of the variable names are pretty intuitive. (e.g., V1_S is the voltage in probe 1 for Survey data).

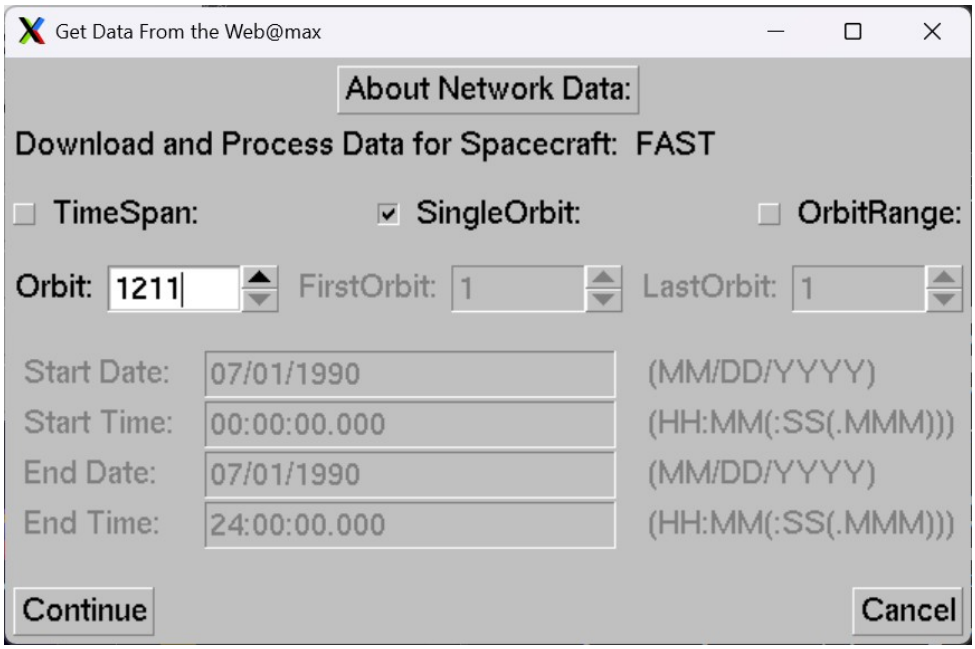
Here we expand the DSP (Digital Signal Processor) menu, and then DSP_Physics, and choose the DQD DspADC_Mag3ac_spectra, which is the on-board measured spectrum for the AC coupled Magnetic field.



We now have a blank plot, TimeSpan Not Set. To set up the time span and plot data, right-click on the window, but not inside the plot, and get the plot menu, note that all options are available. (But since you have not loaded data, most will not work.)

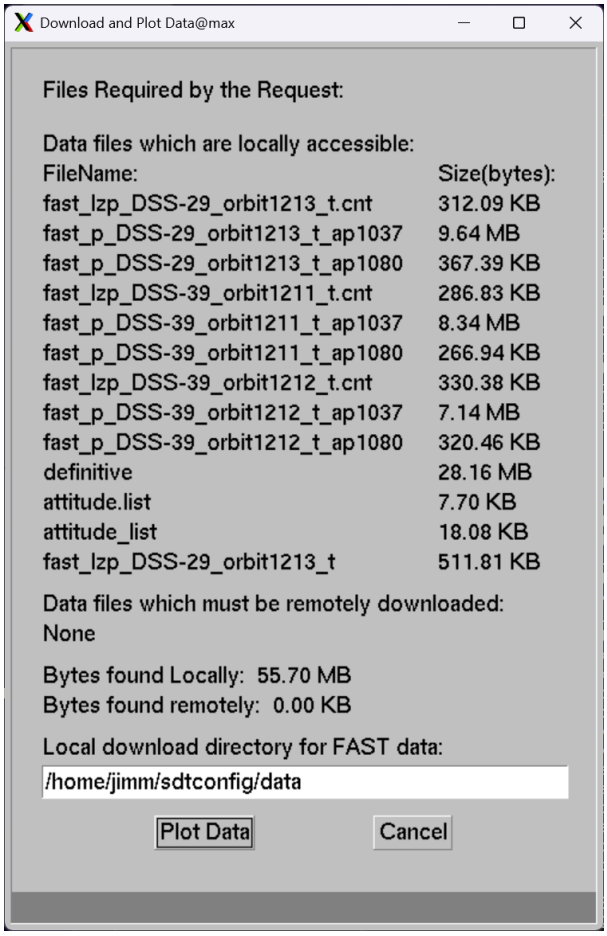


Unless you have a particular data file in mind, it's best to click 'DataFromNetwork' to get the data. Another pop-up.

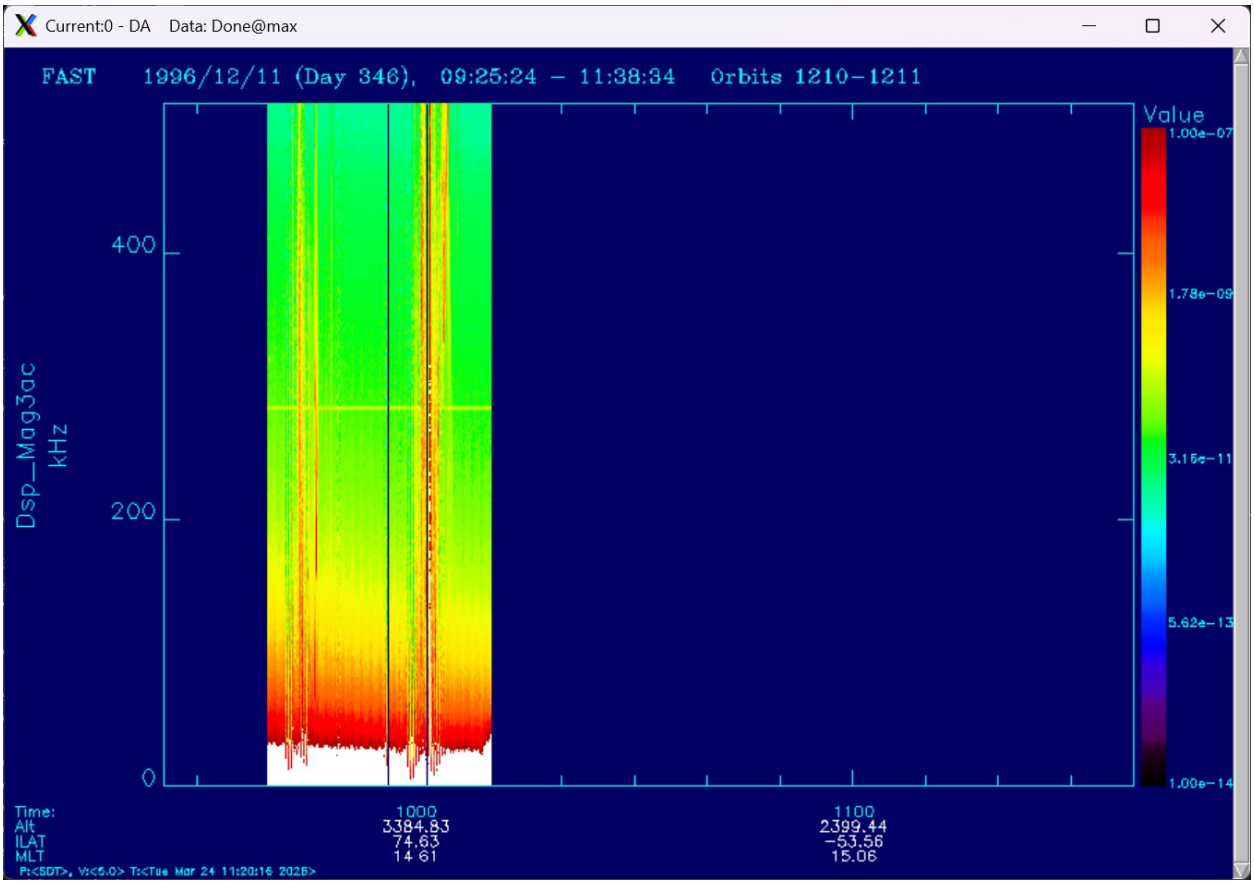


There are options to select a time range, an orbit or range of orbits. We select orbit 1211, where we know we have data. (To check for data, try the FAST summary plot web page, which allows selection by orbit, but not currently by time range. Visit http://sprg.ssl.berkeley.edu/fast/scienceprod/welcome.html, and click on 'View Summary Plots'.)

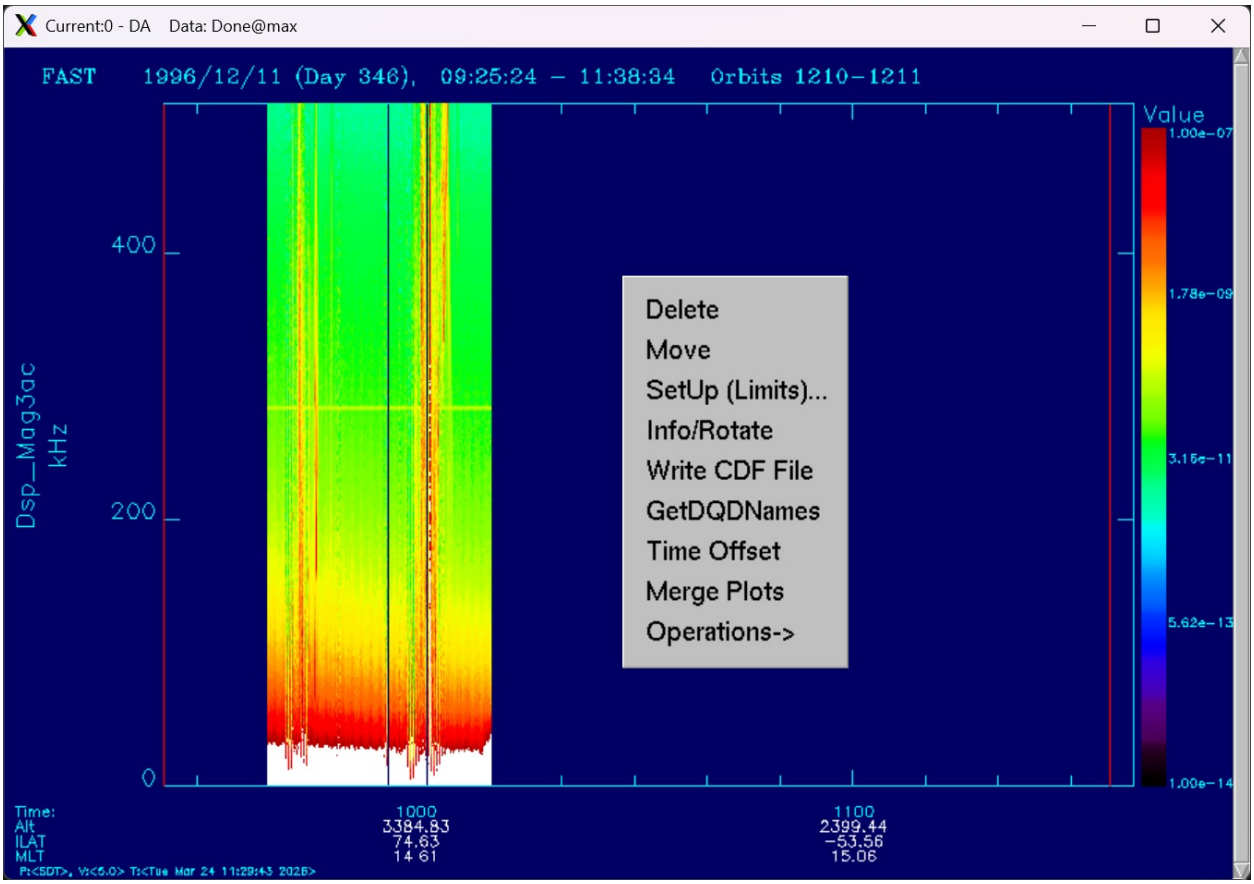
The next SDT pop-up shows the files needed to plot that data.



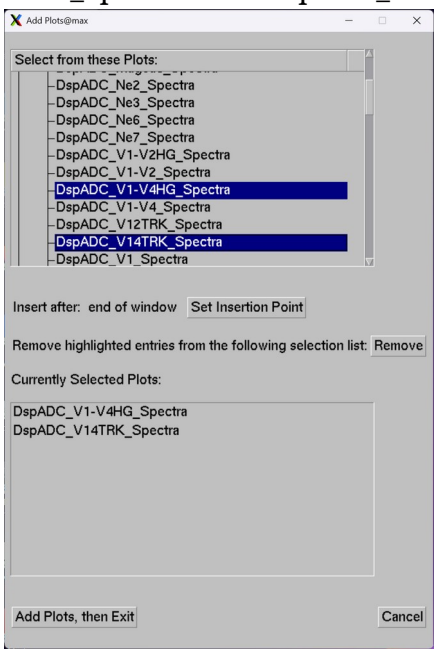
Since here we're working on a local SSL computer, the data is available, and does not need to be downloaded. For remote sites, the files should be downloaded automatically. Click the 'Plot Data' button, and we get the plot.



Plots can be changed or deleted by right-clicking *inside* the plot limits. These controls are pretty intuitive, and we are not going to discuss the options here.



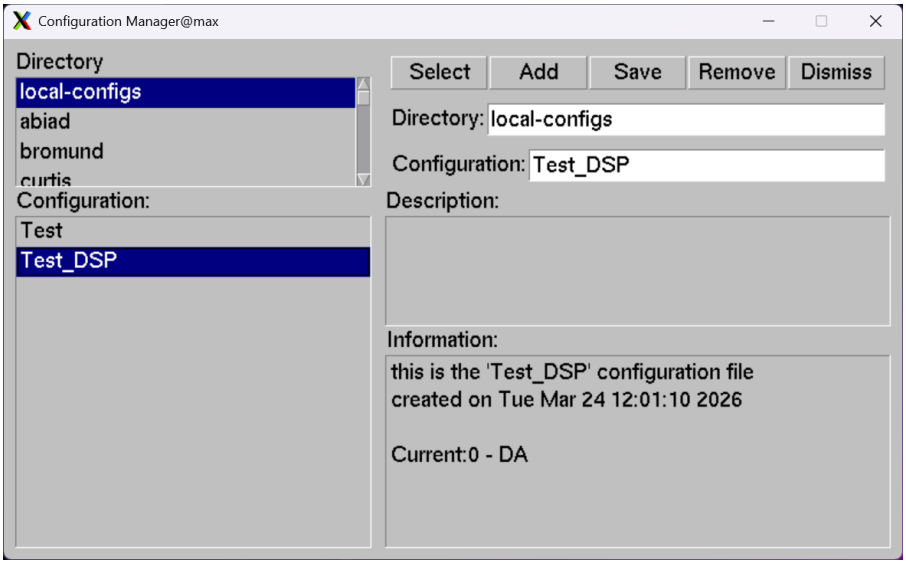
For more data, just right-click on the window (but not the plot), and click the 'Add Plot' option again. You can add multiple plots. Here we add two more data types, 'DspADC_V1-V4HG_Spectra' and DspADC_V14TRK_Spectra', which have available data.



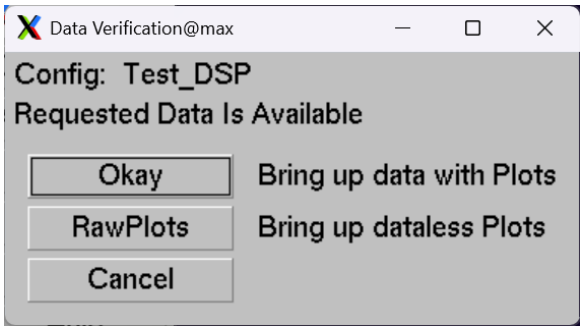
If there is no data for the variable selected, there will be a blank plot.

SDT CONFIGURATIONS:

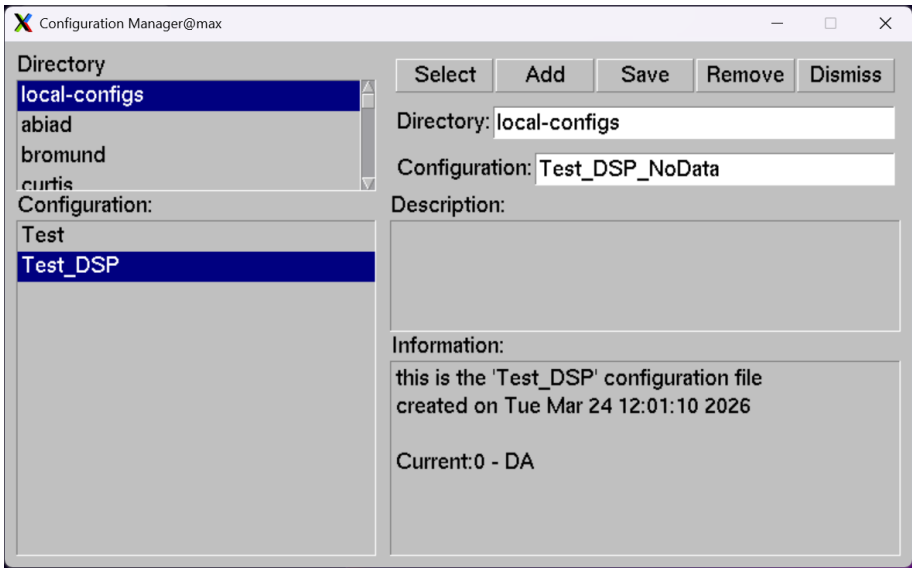
Next, save the configuration. Back on the main menu, click on the 'Configuration' button for the next pop-up. This will have a list of directories, which should be ignored, those are for users from the 1990's -- stick with 'local-configs' to keep everything in the local working directory. Also there will be a list of configurations available below. Here we name our configuration 'Test_DSP', and click 'Save'. A file 'UIcfg.Test_DSP_NoData' will show up in the current working directory. In the next SDT session, this can be loaded, and the plots will show up.



When you load a configuration, you have the option of loading data for the time span in the configuration, or bringing plots without data.



For the batch mode process that we will use to create Data files, we want dataless configurations. To do this, we click 'RawPlots' to get blank plots, and then save the configuration again.



We now have a dataless configuration that we can use for our batch processes.

SDT and IDL:

The FAST EFI L2 process uses SDT and IDL in combination in batch mode, but SDT data can also be accessed from the IDL command line in an interactive session. This is especially useful when debugging processes. All of the IDL code used for FAST processing is available in the SPEDAS distribution (see: https://themis.ssl.berkeley.edu/software.shtml).

If your SPEDAS base software directory is 'my_spdsw', then the FAST mission software can be found in the directory 'my_spdsw/general/missions/fast/'. (Note that there is also FAST software under the my_spdsw/projects/fast/' directory, but this only refers to software for the SPEDAS GUI.)

To run both SDT and IDL then, here we will create and plot a tplot variable with the DSP mag3ac variable:

source setup_sdt

Run sdt in background; select the Test_DSP configuration which will load the data.

sdt &

Then call idl:

idl

In IDL:

To find what data quantities are available:

IDL> dqds = get_dqds()

IDL> print, dqds

The function used to grab FAST fields data is get_fa_fields.pro. First, though, for some FAST quantities need to have a start time defined. Since we know that we have data from orbit 1211, then we can use fa_orbit_to_time.pro.

IDL> orbit = 1211

IDL> orb_info = fa_orbit_to_time(orbit)

The orb_info array contains, the orbit and start and end times. To set a time interval, call timespan:

IDL> timespan, orb_info[1:2]

Then get the data:

IDL> data = get_fa_fields('DspADC_Mag3ac')

IDL> help, data

To plot the data, create the tplot variable (Stored as a Log):

IDL> tvar = 'fa_dspadc_mag3ac'

IDL> store_data, tvar, data = {x:data.time, y:alog10(data.comp1), v:data.yaxis}

Add some plot options (The following steps are hacked from the L2 process function):

IDL> zlim = [-13,-5]

IDL> ztit = 'Log nT!U2!N/Hz'

IDL> ytit = 'LF B 21"!C!C(kHz)'

IDL> options,tvar,'spec',1

IDL> options,tvar,'panel_size',6

IDL> options,tvar,'ytitle',ytit

IDL> options,tvar,'zstyle',1

IDL> options,tvar,'zrange',zlim

IDL options,tvar,'ztitle',ztit

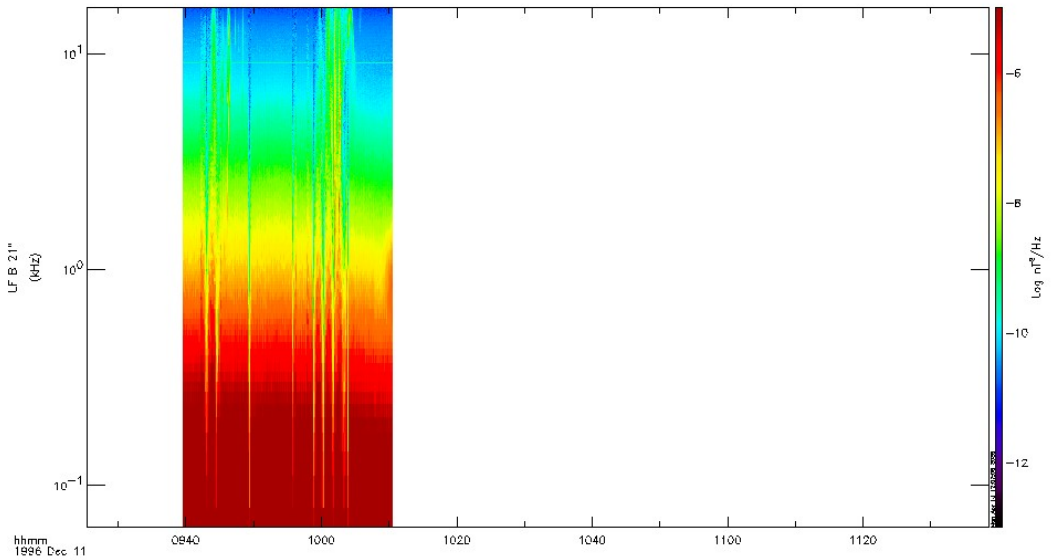
IDL> options,tvar,'y_no_interp',1

IDL> options,tvar,'x_no_interp',1

IDL> ff_ylim,tvar,[0.064,16.384],/log

IDL> tplot, tvar

So there's the data in IDL. The FAST EFI L2 data processing is full of “create tplot variables from get_fa_fields output”. Next we'll describe those programs.



SDT_BATCH:

The FAST L2 EFI processes run SDT using the 'sdt_batch' command. A comprehensive description of the batch command can be found at: https://sprg.ssl.berkeley.edu/~jimm/sdt_stuff/BatchUse.

(If you are local at SSL, the path is /disks/fast/software/forte_6_2/docs/BatchUse.)

The sdt_batch command needs a file that gives the configuration and directs the output. This is our test file Test_DSP.batch, using the configuration Test_DSP_NoData created earlier:

BatchJob: cdftest

Printer: chp260

FileDestination: Test_DSP.ps

Output: ColorPostScript

PlotsPerPage: 8

PageSize: ASize

Orientation: portrait

PageTag: cdf_test

DataOutputFile: Test_DSP_1211 CDF

#NoPlots:

#IDL: run_test_dsp.pro

DataBaseRequest:

Orbit 1211

PlotConfigurationDir: /home/jimm/

PlotConfigurationFile: Test_DSP_NoData

The options are pretty intuitive, BatchJob is an identifier. Printer is a printer. (For L2 processing the NoPlots option is used, so the printer may be fictional.). FileDestination is the name of the output postscript file. DataOutput file is the name of any output cdf files. NoPlots turns off plotting, and is commented out here, so we get a plot file. IDL is also commented out; this option calls an IDL main program that will be able to access the data loaded into SDT. (The FAST L2 EFI processes uses IDL to create the L2 CDF files.) DataBaseRequest can be an orbit number for FAST, or also a time range input. For FAST orbits are more convenient. PlotConfigureaiotDir and PlotConfigurationFile give the configuration. (Note that the configuration filename in the script does not include the 'Uicfg.' prefix on the actual file.)

The next step is to run sdt_batch:

sdt_batch Test_DSP.batch &

This should not take more than a minute. A bunch of files will show up, here is a list:

outdqh

errdqh

outscm

errscm

outUI

errUI

outfast

errfast

SDT is a suite of multiple processes, called 'dqh' (data quantity handler), 'scm' (???), UI (User Interface), and 'fast' (FAST instrument package). The 'out' files are diagnostic output, and 'err' files are error messages. Some typical error messages will be discussed later.

These are the output files:

‘Test_DSP.ps.page1’ is the plot. (Since the printer in this case is non-existent, the plot is not printed, there is an error message in the errUI file.)

There is a CDF file for each data quantity. These have minimal metadata, but can be read using the IDL SPEDAS cdf2tplot command with the /smex_epoch keyword set. Filenames:

Test_DSP_1211.DspADC_Mag3ac_Spectra.cdf

Test_DSP_1211.DspADC_V1-V4HG_Spectra.cdf

Test_DSP_1211.DspADC_V14TRK_Spectra.cdf

To read and plot in IDL:

IDL> filex = ‘Test_DSP_1211.DspADC_Mag3ac_Spectra.cdf'

IDL> cdf2tplot, files = filex, /smex_epoch, varformat = '*'

IDL> options,'DspADC_Mag3ac_Data','spec',1

IDL> options,'DspADC_Mag3ac_Data','zlog',1

IDL> tplot, 'DspADC_Mag3ac_Data'

CREATING FAST EFI L2 FILES:

Now we are ready to create some L2 files. The FAST L2 files are created in a cronjob that calls a c-shell script that creates the batch command file, and runs sdt_batch. This example will be for FAST EFI survey electric field files, which contain despun electric field data from both SDT and IDL, probe voltage values, spacecraft potential, magnetic and sun phase, and coordinate conversion matrices. (Note that all of the shell scripts and batch command files used for FAST EFI L2 file generation are in SPEDAS: 'my_spdsw/general/missions/fast/fa_ops/workdir'.)

Here is the cronjob entry for user jimm (That’s me):

FAST electric field CDf Processing

* * * * * /bin/csh /home/jimm/fast_idl/process_orbit_esv.csh >/dev/null 2>&1

This runs every minute, with no standard output or emails.

The shell script should always be run in its own dedicated directory; here the local directory is '/home/jimm/fast_idl/'. Also run only one job at a time on a given computer, so that the IDL program can find and access the shared memory set up by the SDT process. Otherwise, IDL may pick up the data from the wrong SDT.

Here is the shell script:

```
#!/bin/csh

# Reads in the FAST orbit to process, creates a lock file, writes SDT configuration file, calls SDT_BATCH. The IDL process called by  sdt_batch will handle deleting the lock file, incrementing the orbit number and writing the new ‘orbit.txt’  file.

# Set up IDL path

unsetenv IDL_PATH

source /usr/local/setup/setup_idl8.7.2          # IDL local setup

setenv BASE_DATA_DIR /disks/data/

setenv ROOT_DATA_DIR /disks/data/

#for CDFs

setenv CDF_TMP /home/jimm/data/

#IDL SETUP for SPEDAS

if !( $?IDL_BASE_DIR ) then

    setenv IDL_BASE_DIR /home/jimm/themis_sw

endif

if !( $?IDL_PATH ) then

   setenv IDL_PATH '<IDL_DEFAULT>'

endif

setenv IDL_PATH $IDL_PATH':'+$IDL_BASE_DIR

#Run in /home/jimm/fast_idl

cd /home/jimm/fast_idl

# setup SDT

source setup_sdt
```

```
# Check for Lock file, if it's there nothing happens, note that all  processes all look for the same file, ‘process_orbit.lock’ to avoid piling on processes. Use separate directories.

if (! -e process_orbit.lock) then

# Run an IDL program that kills SDT, if hung (batch mode, no prompt).

   rm -f killsdt0_log

   idl killsdt_all.pro > killsdt0_log

# Kill any remaining SDT processes

   ps -ef | grep 'SDTRunIndex' | grep -v grep | awk '{print $2}' | xargs kill -9

#clean out all CDF files

   rm -f *.cdf

# create lock file

   echo process_orbit_esv > process_orbit.lock

# Check if orbit.txt exists, exit if there’s no file

   if (! -e orbit.txt) then

         echo "orbit.txt does not exist."

         exit 1

   endif

# Read the number from orbit.txt, note the backticks

   set number = `cat orbit.txt`

# Write file for sdt_batch,

   rm -rf process_orbit_esv.batch

   set line="BatchJob:  cdftest"

   echo $line > process_orbit_esv.batch

   set line="Printer:  clf0"
```

```
echo $line >> process_orbit_esv.batch

set line="#FileDestination:  1996_03_06_esa.ps"

echo $line >> process_orbit_esv.batch

set line="Output:   ColorPostScript"

echo $line >> process_orbit_esv.batch

set line="PlotsPerPage: 8"

echo $line >> process_orbit_esv.batch

set line="PageSize:   ASize"

echo $line >> process_orbit_esv.batch

set line="Orientation:  portrait"

echo $line >> process_orbit_esv.batch

set line="PageTag:  cdf_test"

echo $line >> process_orbit_esv.batch

set line="DataOutputFile: orbit_process_$number CDF "

echo $line >> process_orbit_esv.batch

set line="NoPlots:"

echo $line >> process_orbit_esv.batch

set line="IDL: run_process_despin_esv.pro"

echo $line >> process_orbit_esv.batch

set line="DataBaseRequest:"

echo $line >> process_orbit_esv.batch

set line="Orbit $number"

echo $line >> process_orbit_esv.batch

set line="PlotConfigurationDir: SDT_config"
```

```
    echo $line >> process_orbit_esv.batch
    set line="PlotConfigurationFile: ESV3_NoData"
    echo $line >> process_orbit_esv.batch
#Now call sdt_batch using this file
    sdt_batch process_orbit_esv.batch
else #If no lock file,
    echo "process_orbit.lock exists"
#check to see if the lock file is more than 5 minutes old
    set lfile = process_orbit.lock
    set mtime = `stat -c %Y $lfile`   # GNU/Linux 'stat'
    set now   = `date +%s`
# Compute age in seconds
    @ age = $now - $mtime
# Threshold = 300 seconds (5 minutes)
    if ($age > 300) then
            echo "$lfile is older than 5 minutes ($age seconds old). "
            # Run IDL program (batch mode, no prompt)
            rm -f killsdt0_log
            idl killsdt_all.pro > killsdt0_log
# Kill any remaining SDT processes
    ps -ef | grep 'SDTRunIndex' | grep -v grep | awk '{print $2}' | xargs kill -9
#clean out all CDF files
        rm -f *.cdf
#delete lock file
```

```
        rm -f process_orbit.lock
    else
        echo "$lfile is newer than 5 minutes ($age seconds old). Doing nothing."
    endif
endif
```

Here are the steps in the process in non csh language:

0) This process requires a file 'orbit.txt', and a file 'end_orbit.txt'. It will process single orbits, but not if the value in 'orbit.txt' is bigger than that value in 'end_orbit.txt'.

1) Set up the IDL path. This will need to be changed for non-jimm users, to your local IDL SPEDAS setup.

2) Set up SDT.

3) Check for a file called 'process_orbit.lock'. If this file exists, then there is a process running and the script will exit. It is possible that the lock file will be deleted, depending on how old it is. See step (11). If the lock file does not exist, then start a new process.

4) Kill all current hung-up SDT processes. The file creation process should take less than a minute, so if there are any SDT processes running and no lock file, they are hung. When running, SDT creates a file with the process ID numbers of the different processes (UI, dqh, scm). The IDL program killsdt_all.pro reads this file and kill the processes with those Id's. This does not always work, so there is also an explicit kill command that kills any process with the string 'SDTRunIndex'.

5) Clean out all local CDF files.

6) Create a new lock file. Now we can do this orbit.

7) Check for a file called 'orbit.txt'. If it doesn't exist, then exit. If there is an ‘orbit.txt’ file, then read it and process this orbit.

8) Write an input file for sdt_batch, with the orbit number from the 'orbit.txt' file, and the configuration needed to load the data into SDT.

9) Run sdt_batch for the new batch file. The IDL process creates L2 files, increments the orbit number and updates ‘orbit.txt’, and deletes the ‘process_orbit.lock’ file.

10) All done, Exit.

11) If the lock file exists, then the process checks the age of the lock file.

12) If the file is less than 5 minutes old, then do nothing and exit. Maybe SDT is still working.

13) If the file is older than five minutes, then we assume that SDT is hung or has crashed, all SDT's are killed, all CDF's are deleted, and the lock file is removed. The same orbit will be processed the next time that the shell script is called.

Note that there is a possibility for an infinite loop here, but in general, with hung orbits, reprocessing works, and if not, errors are found in the IDL part of the process, and can be debugged in an IDL session.

The IDL output and error messages for the process are found in the files:

outIDL.run_process_despin_esv.pro

and

errIDL.run_process_despin_esv.pro.

If SDT crashes without calling IDL, then there will be a single size 0 CDF file output. If you see this and the orbit does not successfully reprocess, then delete the CDF, and increment the orbit. Here is some code that can be inserted after killing SDT; this was only necessary for SFA processing for a few orbits (see the file in the fa_ops/workdir directory, tmp_process_orbit_sfa.csh):

```
# If there is a single size 0 cdf file, then SDT has no data for this orbit, and will segmentation fault.  Check for a single cdf, and if true, then increment the orbit.txt file

# Get list of .cdf files

        set cfiles = ( *.cdf )

# If there is exactly ONE match, and it really matched (not literal "*.cdf")

        if ( "$cfiles" != "*.cdf" && $#cfiles == 1 ) then

           echo "Found single .cdf file: $cfiles[1], SDT crash"

           set raw = `tr -d ' \t\r\n' < orbit.txt`
```

```
        @ next_orbit = $raw + 1
        rm -f orbit.txt
      echo $next_orbit > orbit.txt
      echo "Updated orbit.txt to $next_orbit"
       endif
#clean out all CDF files
    rm -f *.cdf
#delete lock file
    rm -f process_orbit.lock
```

14) Exit.

Now let’s discuss the IDL program.

IDL FOR L2 FILE CREATION FOR DESPUN ELECTRIC FIELD SURVEY ‘ESV’ DATA.

The IDL main program is called from the batch process:

sdt_batch process_orbit_esv.batch

which in turn calls:

run_process_despin_esv.pro

which is a wrapper for the IDL procedure fa_despin_process.pro, that creates despun electric field variables and writes to the L2 files. Here is a step-by-step description of the process:

1) Read the 'orbit.txt' file. If there is no file, then the process will eventually crash.

2) Read the 'end_orbit.txt' file. If there is no file, then the end_orbit variable is set to 51000, although there is no data beyond orbit 24635, so maybe someday reset the end_orbit.

3) If orbit > end_orbit, then Return.

4) Check the datatype, which can be input via the 'datatype' keyword; the default is datatype='esv', for EFI Survey data. Datatype can also be set to 'e4k' and 'e16k', for 4K and

16K burst data. (Datatypes 'sfa' and 'dsp' are processed using an IDL program fa_spec_process.pro.)

5) If the keyword(no_overwrite) is set, then if there is a file for this orbit, Return.

6) Next, despin the data, this calls the program fa_fields_despin3.pro, which we will discuss in the next section. After despin, check to see if the process was a success. The despin process for survey data produces two tplot variables 'fa_e_near_b', and 'fa_e_along_v'. If these variables have no data, then print a message, but continue.

7) Call get_fa_potential.pro to get variables for spacecraft potential and spin-averaged potential. If the potential variables have no data, then print messages.

8) Set up arrays of variables and SDT DQD names to test. The string array ‘to_be_stored’ contains the tplot names of the data variables that will be stored in the L2 CDF. The array ‘to_check’ has the tplot names of variables to check to be sure that there is data to store. (This is a subset of the ‘to_be_stored’ array.) The array ‘dqds_to_check’ contains the SDT DQD identifiers to check for data. For a given DQD to be found, there should be a CDF file for that DQD that is larger than the value of the variable dqd_size_limit (deafult is 5000 bytes, but maybe should be 6500). Each datatype has its own set of tplot variables and DQDs to check.

9) Do variable and file tests. If the program cannot find data for ‘to_check’ and ‘dqds_to_check’, then the process fails, and no L2 file is created. Also the orbit number is added to a file in the local working directory called 'orbits_no_data.txt'. This is to aid in reprocessing; most of the time, if SDT or IDL fails, then there is no useful data.

10) If there is no data, then increment the orbit number, write the new orbit to 'orbit.txt' file, remove the 'process_orbit.lock' file and return. We are done.

11) If the keyword 'full_database_management' is set (this is always true), then the next step is to set up the CDF global attributes, create a filename ('fa_despun_'+typ+'_l2_'+date+'_'+orb_str+'_v02'), and set up the output directory ('/disks/data/fast/l2/'+orbit_str, where orbit_str is the orbit number mod 1000 padded to 5 digits, e.g., 02000 for an orbit from 2000 to 2999).

12) For each tplot variable in the ‘to_be_stored’ array, call fa_despin_process_vatt.pro which creates the CDF variable attribute structure for that variable. This is where most of the messy CDF attribute work is done. Then call the program fa_tplot_add_cdf_structure.pro, which adds a default variable attribute structure to the

tplot variable. The default structure is then replaced by the output of fa_despin_process_vatt.pro.

13) Once all the variables have their attributes, then call fa_tplot2cdf.pro to write the file.

14) Increment the orbit number, write the new orbit to the 'orbit.txt' file, remove the 'process_orbit.lock' file and return. All done.

DESCRIPTION OF FAST_FIELDS_DESPIN3:

The SDT data quantities are pushed into tplot variables in the IDL procedure fast_fields_despin3.pro. This is an extension of the original fast_fields_despin.pro from 1996. The despin process creates the spin-plane electric field variables, 'fa_e_near_b' and 'fa_e_along_v'. The variable 'fa_e_near_b' is the survey mode spin-plane electric field, in the direction corresponding to zero degrees magnetic phase, as described in http://sprg.ssl.berkeley.edu/fast/scienceops/fast_fields_help.html. The variable 'fa_e_along_v' is orthogonal, representing the electric field in the direction corresponding to ninety degrees magnetic phase. Here are the steps:

1) First determine if there is data. The SDT DQD's required are 'V5-V8_S','V1-V2_S', and 'SMPhase_FieldsSurvey0', which represent the voltage difference for probes 5 and 8, the voltage difference for probes 1 and 2, and the magnetic field phase.

2) Prepare data for despin; call get_fa_fields.pro and fa_fields_phase.pro, to get the v12 and v58 potential differences and phase data into IDL structures, and call fa_fields_combine.pro to interpolate all of the needed quantities to the v58 time array.

3) Check for 'notches', using ff_notch.pro. Fields during 'shadow_notches' (or 'shadow spikes' or 'sun spikes') are set to NaN. Magnetic notches, or pulses, are removed and replaced with interpolated values.

4) Use ff_quickfit.pro to spin-fit the data. This procedure takes each data quantity (v12 and v58) and returns a zero level and sine and cosine values for the spin fit.

5) Use ff_dce_fix.pro to subtract the zero level from the spin fit of v58 data.

6) Calculate the ratio of the totaled spin fit values of v58 to v12.

7) Use ff_dce_fix.pro again, to subtract the zero level from the spin fit of v12 data, and then adjust the v12 gain by multiplying the v12 data by the ratio of v58/v12 spin fits from the previous step.

8) Next, despin: e_near_b = v12*cos(Bphase+dphi) - v58*sin(Bphase+dphi), where bphase is the magnetic phase, and dphi is the angle between the 1-2 boom and the magnetic field axis. (See https://sprg.ssl.berkeley.edu/fast/scienceops/docs/fast_fields_description.pdf).

9) e_along_v = v58*cos(Bphase+dphi) + v12*sin(Bphase+dphi)

10) Store the despun field, magnetic phase and sun phase angles into tplot variables.

11) Store tplot variables for boom pair length and phase. *(These are not used or output, so this step may be deleted.)*

12) Create quality flags for 'notches', stored bit-by-bit: 0 for ok data, first bit set for 12 mag notch, 2nd bit for 12 shadow notch, 3rd bit for 58 mag notch, 4th bit for 58 shadow notch.

Only ESV, survey, data files contain quality flags. Also, only the ESV, 4K and 16K L2 have the despun field output, calculated via IDL. The SFA and DSP L2 files are all direct SDT output, as can be seen in fa_spec_process.pro. The following saved survey data quantities are all direct SDT output.

13) Store direct SDT output in tplot variables. Spin plane fields (E_0_S) are saved in GSE, GSM and DSC (Despun spacecraft coordinates). All available voltage differences and probe voltage values are stored. Coordinate conversion matrices from DSC to GSE and GSM coordinates are stored, in addition to spin axis direction coordinates. The list of tplot variables created in fa_fields_despin3.pro and saved into the L2 file is:

'fa_e_near_b','fa_e_along_v', 'fa_e12','fa_e58','fa_bphase', 'fa_sphase', 'fa_e0_s_gse', 'fa_e0_s_gsm', 'fa_e0_s_dsc', 'fa_v1_v2_s', 'fa_v1_v4_s', 'fa_v5_v8_s', 'fa_v9_v10_s', 'fa_v2_s', 'fa_v4_s', 'fa_v6_s', 'fa_v7_s', 'fa_v8_s', 'fa_v9_s', 'fa_v10_s', 'fa_dsc_gse', 'fa_dsc_gsm', 'fa_spin_axis_gse', 'fa_spin_axis_gsm'

('fa' for FAST, '_s' for Survey data. 'fa_e12' and 'fa_e58' are the non-despun spin plane data.)

REPROCESSING MISSING ORBITS:

If, for some reason, a a file for a given orbit did not show up, and the orbit number is not included in the 'orbits_no_data.txt' file, then it's easy to reprocess the orbit by editing the

'orbit.txt' file, and calling the shell script from the linux command line, /bin/csh process_orbit_esv.csh. If there is an issue that causes a segmentation fault error, then try a different computer; it is possible that some process interfered with the shared SDT-IDL memory, and a different computer might work.

If reprocessing still does not work, then check the ‘errUI’, ‘errscm’, ‘errdqh’, ‘errfast’ files for error messages.

Another option exists; if there is no time to sit and run the script by hand, then there are scripts for reprocessing, e.g., reprocess_orbit_esv.csh. The reprocess script is the same as the process script, except it checks for an L2 file for the input orbit. If there is a file, then no process, but the 'orbit.txt' file is updated. Generally, the reprocess will pick up any data that failed for segmentation faults or network glitches.

For the L2 processes for the 24635 orbits, for each of the datatypes, 'esv', e4k', 'e16k', 'sfa', 'dsp', the initial process failed for 100 to 200 orbits. After reprocessing most missing files filled in. A summary of orbit numbers, for which each of the processes failed, with comments regarding error messages from IDL or SDT, can be found and https://sprg.ssl.berkeley.edu/~jimm/sdt_stuff/missed_orbits_with_data.txt.

CHANGING CDF ATTRIBUTES:

So now we have files, but these may not be ISTP-compliant. Here at SSL, thanks to Jim Lewis (jwl@ssl.berkeley.edu) we have the ability to change global and variable attributes in CDF files without reprocessing the data. This saves an enormous amount of processing time. The commands live under the /disks/socware directory, and can be accessed using the following setup:

source /disks/socware/thmsoc_2_00_production/src/tmtools/tmtools_setup.csh

Then the commands cdf_gattr and cdf_vattr are used to change the attributes in the CDF file, without changing the data:

cdf_gattr filename "New global Attribute"

cdf_vattr filename cdf_variable attribute "New Variable Attribute"

To do for multiple files, put the commands in a shell script, then call find for CDF's in the file system, and run the output from the find command into the attribute shell script.

Here is an example:

Create a file: ‘fast_esv_att_change.ksh’ (This is a KSH script, similar to a bash or csh script. We use these for THEMIS processing.)

```
#!/usr/bin/ksh

#FIle: fast_esv_att_change.ksh

#One-time change of attributes for all FAST L2 ESV files:

#One argument input, full path filename

# first source tmtools_setup.csh, prior to calling the script.

#set the argument to filename

esv_cdf_file=${1}

# Say I want to change the logical source description attribute:

cdf_gattr ${esv_cdf_file} Logical_source_description "Spacecraft Collected Survey Electric Field (ESV)”

# And maybe a variable attribute:

cdf_vattr ${esv_cdf_file} fa_e_along_v CATDESC “New Catdesc for FAST despun E_ALONG_V”

# Add as many commands as needed...
```

Next run the command to do this for all files, find all filename and execute the command.

```
#!/usr/bin/ksh

find /disks/fast/data/l2/esv -name "*.cdf" -exec fast_esv_att_change.ksh {} \;
```

That’s all. I hope this helps.

Jm

jmctiernan022@gmail.com