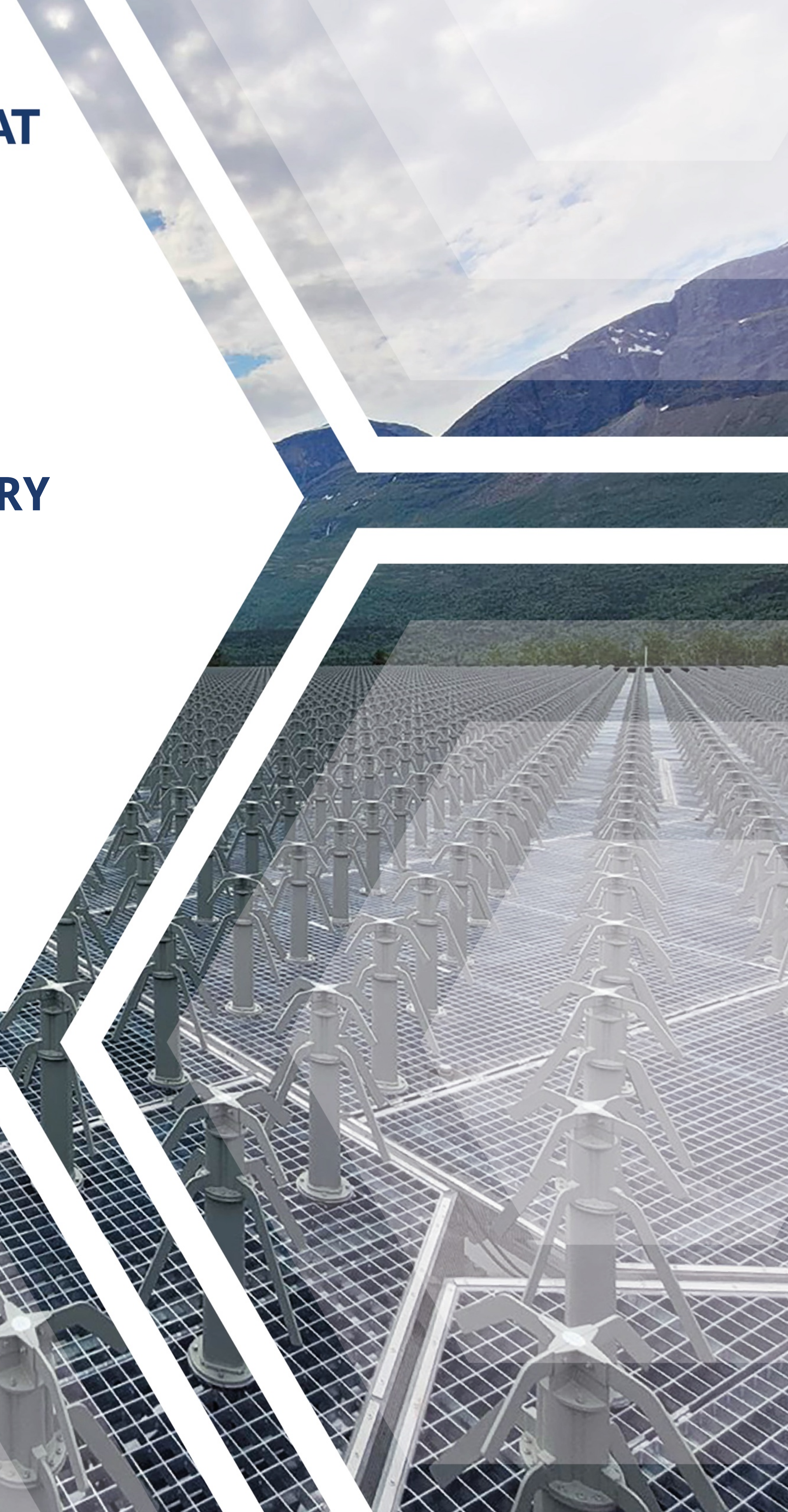
EISCAT

DSP LIBRARY

www.eiscat.se



Make your own “type”

It is now possible to make your own data handler for special experiment where you want to do some data processing which is not all ready available in some other fil type.

In this example we will make a simple zero lag calculator. The calculation engines are implemented as on demand loadable libraries for the decodump process on the local server. For a given experiment you can have several loadable libraries, and inside each library you can have several subroutines. Here we will only use one library and one subroutine.

Our new type that we will make, takes the raw data vector and make one integrated zero lag height profile. In this example we are using the tau2pl for the remotes.

The fil file will look like this. As seen we have added a type=6 which are used for all our new calculation types.

What we need to define is the number of data point taken on the channel board in each ipp (vec_len=128;)

- where data starts in the channel boards memory (data_start=0;)
- how many data points it will be in the saved Matlab data file (nr_out_points=128;)
- how many data ipp in each integration period (nr_rep=896;) We have 64 STC in the tlan file and for a integration time of 5 s we need to do 14 loops 64*14=896
- we are sending raw data and do the data manipulations in the server, of course (send_raw=1;)
- the name of the library used (user_lib_name=tau2pl;) this is a shared library file which must be found at the same place as the fil file,
- the name of the subroutine inside the shared library (user_subr_name=zero_lag;),
- number of integer parameters that are needed inside the zero_lag subroutine to do the calculations (dpar_len=2;)
- also floating point values can be send too, in this case we have three parameters (fpar_len=3;)
- the file name where the parameters are defined (par_file=`tau2pl_par.txt`;). Note in this file the integer parameters need to come first, after these the floating point values are defined if used.

The source file for this library is found here, the comments inside this file explain the very simple software interface. The file name is the name defined as the user_lib_name, and the subroutine name is defined with the user_subr_name in the fil file. To compile this C source file and get the loadable library one can use this Makefile. Note this make file can only be used with GNU's version of make.

Another and more useful example is to do some post processing on the time sliced data from the arc_dlayer_ht experiment. This D-layer experiment have a ~0.2 s time resolution and are mainly used to look at heater induced phenomenas on PMSE. This experiment gives every 2 s a Matlab dump containing 10 time sliced lag profile for the D-layer and E-layer. For RTG and quick looks some means of integrating the individual time sliced lag profiles are needed. This show a way to have these quick look lag profiles already in the Matlab data dump. Here you can find the used arc_dlayer_ht_int.fil file.

As a background we need to know that decodump will process data as described in the .DECO file line after line, this fact is here used. We have added a dummy channel in this case channel three. This channel is defined at the end of the fil file and thus will be in the end also of the DECO file. We have set vec_len=0 in all type=6 for channel=3 just to have a place holder, so no data will be read from channel board number three. This is our loadable library routine (arc_dlayer_ht.c) which will take care of integrating our time sliced lag profiles. The crucial thing to know about that we are using in this case is that we have access to the full current input and output data vectors, with current it means all data defined above in the DECO file can be reached. The pointers which are fed into the user defined subroutine is the current data positions as defined in the DECO file, so if we want to use older data from the raw data input or the output data already made we just jump backwards in these data vectors.

1. A Pthread example

Some more programming is necessary to use pthreads and symmetric multiprocessing (SMP) which is needed when more complicated decoding algorithms are implemented. In a SP for studying the D-region at ESR with complementary codes the decoder for the real time graph (RTG) is using pthread to shorten the decoding time. This experiment is using a 256 bits complementary code and the clutter subtraction is using a clutter profile from the current integration cycle. Also pre integration is used. The decoder it self is a ordinary FIR filter. Also this user defined library can save decoded data and a background and noise injection power profiles to text files, this option will not be studied in great detail, but the code it self is still in the source for the library, also a subroutine for simulating a hard target is included.

First a clutter profile is made for each of the two code sets, also these profiles are scaled. This is done in a single thread. Next is to do a coherent pre-integration and decoding for each code set.

Two threads are used one for each code set. When the pre-integrated decoded data for each code set is done, the two set of data is summed together for each IPP, also done in a single thread, all possible lags are calculated and stored. The parameter file contains as usually the parameters needed for for data handling like number of samples, number of loops, and the used code sets. The jurg3.c is the C library which have the main jurg3_complementary_deco subroutine something to note is that user parameters which are set from EROS is used to toggle certain functions inside the user supplied library like clutter subtraction and simulation of a hard target, definitions and usage of these upar's can be found in jurg3.elan.

A good book to read more and learn about Pthreads is "Pthreads Programming" by Bradford Nichols and Jacqueline Proulx Farrel, O'REILLY ISBN 1-56592-115-1

Please note that these examples just show some possibilities with the shared libraries, the first example are very naive and in real life a type=1 should be used. The main problems to deal with is the performance and stability, at all EISCAT sites we have multiprocessor servers, and to use them in a efficient way you need to make your subroutines to use as many of the available processors in parallel as possible. In the decodump program the POSIX pthread interface is used. Also your own subroutines needs to be carefully checked with some type of runtime debugger. A possible way to do this is to develop your subroutines on a LINUX machine, make a small wrapper program where you can supply known data and examine the output and check for errors. Also use a runtime checker for memory overwrites, threading errors and such. A very good runtime checker for LINUX is the Valgrind debugger. If your subroutine contain errors it will crash the decodump process, so be care full when developing.