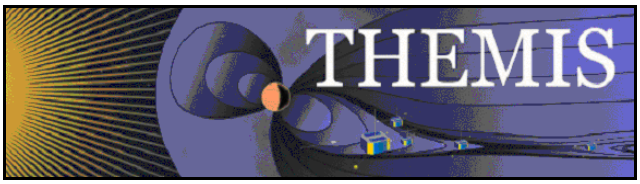


THEMIS
Science Data Analysis Software (TDAS)
Developers Guidelines

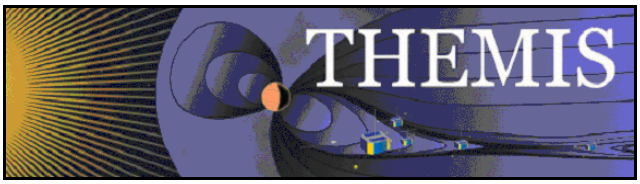
THM-SOC-128
November 2011

Pat Cruce, THEMIS Science Software Engineer

Lydia Philpott, THEMIS Science Software Engineer

David King, THEMIS Science Software Manager

Vassilis Angelopoulos, THEMIS Principal Investigator



Document Revision Record

Rev.	Date	Description of Change	Approved By
-	3/25/2010	Initial release - Pat Cruce	D. King
1	11/28/2011	Enhancements - Lydia Philpott	D. King

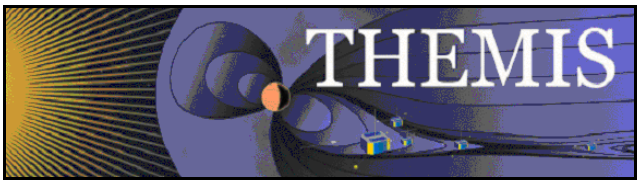
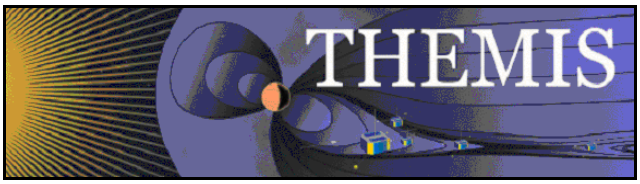
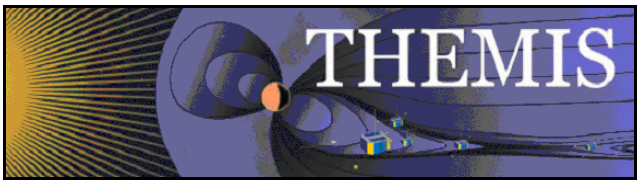


Table of Contents





1. Purpose and Scope

The THEMIS Science Data Analysis Software (TDAS) is a suite of scientific analysis routines built in IDL. The main function of the software is to allow users to download and read THEMIS related data in CDF format, perform analysis on this data, and produce publication quality plots. TDAS offers both a command-line interface and a graphical user interface. The IDL routines are derived from those used by the Cluster, Wind, Polar, and FAST missions. TDAS makes extensive use of TPLOT: a data plotting and management tool. The use of TPLOT variables in TDAS is discussed in Section 2. Section 3 briefly discusses THEMIS CDFs. The remainder of the document covers general programming style guidelines. Additional THEMIS documentation can be found on the document ftp site:
ftp://apollo.ssl.berkeley.edu/pub/THEMIS/3%20Ground%20Systems/3.2%20Science%20Operations/Science%20Operations%20Documents/.

TDAS is designed to be of general use, applicable not just to THEMIS data quantities. The software, for example, has routines for loading GOES, WIND, and ACE data, along with data from a range of ground based magnetometer networks. This document is intended for external developers developing software for inclusion in TDAS.

2. TPLOT

2.1 Introduction

At their most basic, TPLOT variables, as implemented by “get_data” and “store_data”, are really just a map where a key-string is associated with three arbitrary data structures of any IDL type. To guarantee that TDAS routines can properly use these TPLOT variables we maintain a set of naming and structural conventions for each component of a tplot variable.
The following command shows each component of the tplot variable as used in the store_data command:

```
store_data,'varname',data=data,dlimits=dlimits,limits=limit
```

The following four sections, varname, data, dlimits, limits, will describe each of these components.

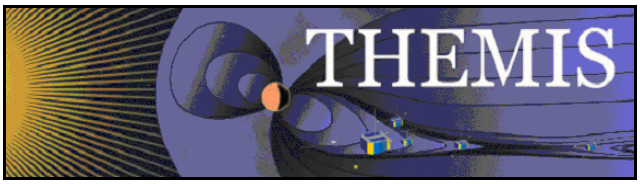
As a caveat, note that these are conventions we try to follow in writing new code, but they are not always consistently adhered to in older routines. As development continues we are working to make the whole code base more consistent and standards compliant.

2.2 Variable Names

This value can be any string that does not contain the characters: ' '(space),'*'(asterisk),'?'(question-mark),'['(left-bracket),']'(right-bracket),'\'(backslash). By convention, TPLOT names do not include other extraneous punctuation like parentheses and periods, but underscores are used to delineate separate fields in a name. The generic naming convention is normally (source/site/spacecraft)+'_'+(instrument/category)+'_'+(datatype/specific type). For example: "tha_state_pos" contains the state data (specifically position) for THEMIS probe a.

Detailed descriptions of the specific naming conventions used by THEMIS can be found in the document "thm_soc_105_FIELDS_VARNAMES_YYYYMMDD.pdf". This document is available on the THEMIS website, http://themis.ssl.berkeley.edu (currently: thm_soc_105_FIELDS_VARNAMES_20060929.pdf).

It is strongly recommended that all data being loaded by the TDAS software follow some sort of consistent and unique naming convention so that data can be found programmatically using globbing string searches (i.e. pattern matching).



2.3 Data Components

The data component of a tplot variable can be either a struct or a space-separated list of existing tplot names.

2.3.1 Struct

The basic type of tplot data component is a struct-based component. At a minimum it requires two tags, 'x' and 'y', it can optionally have a tag called 'v' or two tags called 'v1' & 'v2'.

2.3.1.1 x-component

a. The x-tag should be bound to the time-array associated with this variable.
b. The time-array should be a one dimensional, double precision array with N elements, where N is the number of time samples in the data quantity.
c. Each element in the time-array should be a unix time (an offset in unleaped seconds since 1970-01-01/00:00:00).
d. The time-array does not need to have a constant cadence (i.e. the difference between adjacent elements can vary from element to element).
e. The time-array is required to be ascending (the i+1th element should always be greater than the ith element).

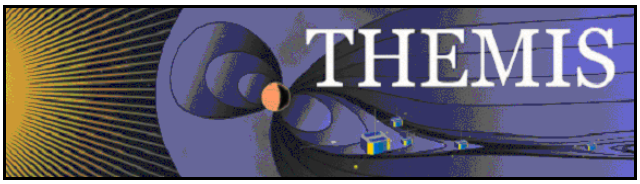
2.3.1.2 y-component

a. The y-tag should be bound to the data values sampled at each time.
b. It can be a one, two, or three dimensional array of any numeric type (complex type is not consistently supported throughout TDAS).
c. The first dimension of the y-component should always have N elements where N is the number of elements in the time-array (x).
d. The second dimension is optional and is used to group data with the same times. In the case of 3-vectors, the y-component should be an Nx3 array. In the case of spectral data, the y-component should be an NxM array, where M is the number of channels, frequencies, energies, etc... in the data.
e. A third dimension can also be added with L elements. The third dimension is generally used when a two dimensional data point is collected for each time. For example, an MxL image may be collected at each time sample N, or a time-varying rotation matrix may be represented as an Nx3x3 array (N x Column x Row).
f. TPLOT variables with 3-dimensional y-components are not always handled consistently by TDAS routines, and often have a special collection of routines to deal with that particular type of 3-d data.(for example time-varying rotation matrix routines are in ssl_general/cotrans/special)

2.3.1.3 v-component

The v-component is an optional component appropriate (but not required) when the y-component is two dimensional.

a. The v-tag should be bound to the scaling data for the M component of a variable with a two dimensional y-component.
b. The v-component is optional. If not-present, it is assumed that data is spaced evenly (equivalent to a v component equal to dindgen(M)).
c. The v-component is used to determine the spacing between the components along the non-temporal axis when transforming and plotting the data. This can be considered a second independent variable (after time).
d. The v-component can be a one or two dimensional array of any numerical type (complex type is not consistently supported throughout TDAS).
   - A one dimensional (M element) v-component indicates that scaling is constant across time. For example, an FFT may have the same frequencies associated with frequency bins at each time.



- A two dimensional (NxM element) v-component indicates that scaling varies across time. It will contain scaling values for all M-channels at each time in N. For example, a particle instrument may be configured to have 16 energy bins at one set of times, and 32 energy bins at the times after a mode change. In such a case, the v-component would have Nx32 elements, and during the times when there are only 16 valid energy bins, the last 16 bins of the array would contain NaNs.

2.3.1.4 v1/v2-components

The v1/v2 components are optional components, appropriate when the y-component is three dimensional.

a. The v1/v2 tags should be bound to the scaling data for the M & L components of a variable with a three dimensional y-component.
b. The v1/v2 components are optional, but if one is present, the other should be present as well.
c. Interpretation of the v1/v2 components and dimensional restrictions varies depending upon the routines used to process/plot the data.

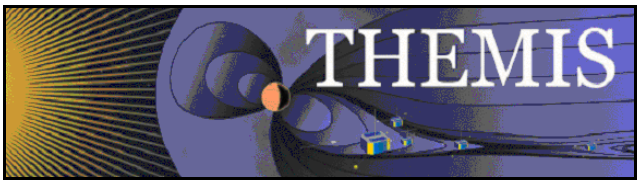
2.3.2 List of TPLOT Names

A data component composed of a list of space separated variable names is called a pseudo-TPLOT-variable or a multi-TPLOT-variable. This convention is used to bind data quantities that have different dimensions together to indicate that they should all be plotted on the same plot. Data processing routines generally do not accept pseudo-variables as inputs. Settings that apply to all names listed in the pseudo-variable may be stored in the dlimits/limits of the pseudovariable, rather than in the dlimits of its individual component variables.

2.3.3 Data examples

Examples of all the data-forms using dummy data and names follow:

```
store_data,'varname1',data={x:dindgen(100),y:dindgen(100)}
store_data,'varname2',data={x:dindgen(100),y:dindgen(100,3)}
store_data,'varname3',data={x:dindgen(100),y:dindgen(100,15),v:dindgen(15)}
store_data,'varname4',data={x:dindgen(100),y:dindgen(100,15),v:dindgen(100,15)}
store_data,'varname5',data={x:dindgen(100),y:dindgen(100,200,300)}
store_data,'varname6',data={x:dindgen(100),y:dindgen(100,200,300),v1:dindgen(200),v2:dindge
   n(300)}
store_data,'varname_pseudo',data='varname1 varname2'
```



2.4 dlimits

The dlimits struct is used to store metadata (data that describes the data). The data in the dlimits is, by convention, set only by load routines and data processing routines. The dlimits has three components, the CDF substruct, the data_att substruct, and default plot settings.

2.4.1 CDF substruct

The cdf substruct is a component generated automatically by cdf2tplot (and routines that use cdf2tplot). It should be considered read-only by all routines after load.

a. The cdf substruct has the following tag_name/signature: dl={cdf:cdf_substruct}, where cdf_substruct indicates a variable pointing to the structure itself.
b. The cdf substruct contains four elements: filename,gatt,vname,&vatt.
   - filename is a string naming the cdf from which the variable was loaded
   - vname is a string naming the variable that was loaded from the cdf
   - gatt is a sub-struct containing all the global attributes in the cdf
   - vatt is a sub-struct containing the variable attributes from the cdf that are specific to the loaded variable.

2.4.2 data_att substruct

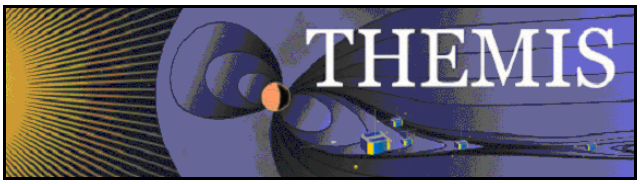
The data_att substruct contains non-plottable attributes of the data. These are used for informational purposes and for allowing various data processing routines to make various automatic decisions.They are normally set initially by load routine post-processing functions, but they may be modified by data processing routines and the user as necessary.
data_att should contain the following three elements, if applicable: units,coord_sys,st_type

a. units: A string containing the units for the variable. This string should not contain any additional formatting, but should include any SI prefixes that are relevant (i.e. list units as nT, not T). Units use the following symbols to indicate operators in compound units:
   - '/' slash: division
   - '(' left-parenthesis: open grouping
   - ')' right parenthesis: close grouping
   - '^' up-carot: exponentiation
   - '*' asterisk: multiplication.

   The strings 'unknown' or 'none' may also be used.
b. coord_sys: string that contains a short identifier to identify the coordinate system of the data. Valid names include: 'spg', 'ssl', 'dsl', 'gei', 'gse', 'gsm', 'sm', 'geo', 'sse', 'fac', 'mva', 'rxy','sel'. Definitions of many of these coordinate systems can be found in the document: thm_soc_110_COORDINATES_20100729.pdf (check for most recent version of this document at the THEMIS document ftp site)
   If one of these coordinate systems does not accurately describe the coordinate system of data, it is better to create a new identifier, rather than use one that does not fit. Alternatively, the strings 'unknown' or 'none' may be used.
c. st_type: string that indicates whether a quantity is a position, velocity, or neither. This is required to properly transform some quantities into different coordinate systems. Valid values are 'pos', 'vel', or 'none'
d. Other informational components may be included in the data_att like 'project', 'instrument', 'observatory', 'filename', but these values are not required for transformations.

2.4.3 Plotting options

Plotting options exist at the top level of the dlimits struct. They are commands and settings to the tplot & tplot_gui routines, which can be set using the 'options,/default' routine or by storing the dlimits struct directly.
Common components include:



a. spec: 1 Indicates that a quantity should be plotted as a spectrogram.
b. x/y/zrange: [0D,1D]: Indicates the data range.
c. x/y/zlog: 1: Indicates the axis should be logarithmic.
d. labels: ['label1','label2','label3']: Indicates y-axis labels.
e. colors: [0,1,2]: Indicates color indexes for the data.

Additional values/examples can be found by consulting the TPLOT cribs in themis/examples/tplot_cribs/
Plotting options can also be placed in the dlimits struct of a pseudo-TPLOT-variable

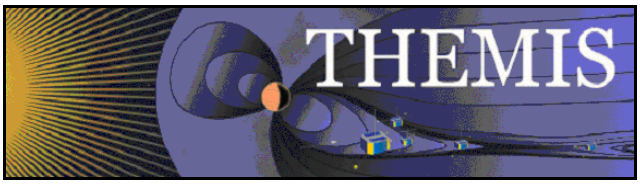
2.4.4 dlimits examples

Examples of hypothetical dlimits storage operations:

```
store_data,'varname1',dlimits={cdf:cdf_struct,data_att: $
    {coord_sys:'gse',st_type:'pos',units:'km'},colors:[2,4,6],labels:['X','Y','Z']}

store_data,'varname2',dlimits={cdf:cdf_struct,data_att:{coord_sys:'none',st_type: $
    'none',units:'eV/(cm^2*sec*sr*eV)'},spec:1,ylog:1,zlog:1,ysubtitle:'eV/cm!U2!N!C-s-sr-eV'}
```

2.5 limits structure

The limits structure contains the same types of plot settings that are in the dlimits, but it can and should be modified by individual user commands during an interactive plotting session. The limits doesn't contain a data_att or cdf component, and can be 0 to indicate no limits have been set. TPLOT settings in the limits structure will override settings in the dlimits.



3. CDF

3.1 Introduction

THEMIS CDFs generally follow ISTP and SPDF guidelines for CDF construction. Detailed documentation can be found on their website (http://spdf.gsfc.nasa.gov/sp_use_of_cdf.html). Examples can also be found in TDAS routines ssl_general/CDF/cdf2tplot.pro and themis/common/thm_load_xxx.pro, as well as in the various load routines in the TDAS distribution. For extensive documentation of the CDF files in the context of THEMIS, see the Level 1 and Level 2 File Definitions documents. THEMIS CDFs contain two significant deviations from the standard, discussed below.

3.2 DEPEND_TIME

All CDF variables that are loaded into our software via cdf2tplot.pro require an attribute called DEPEND_TIME that specifies the name of a CDF variable in the same file which contains UNIX times. UNIX times are double precision offsets in unleaped seconds from the UNIX epoch: 1970-01-01/00:00:00.

3.3 DEPEND_0

ISTP requires a DEPEND_0 attribute on time-varying variables. The value of the DEPEND_0 attribute must be a variable name in the same CDF, with type CDF_EPOCH or CDF_EPOCH16. For THEMIS CDFs, this variable will be marked VIRTUAL, and will not contain any values. It will have a "FUNCT" attribute specifying the name of an IDL function, and COMPONENT_0 and COMPONENT_1 attributes specifying the arguments to that function, which can be used by SPDF to fill in the CDF_EPOCH or CDF_EPOCH16 timestamps from the DEPEND_TIME variable and epoch. Default THEMIS values for "FUNCT" are "COMP_THEMIS_EPOCH" or "COMP_THEMIS_EPOCH16"

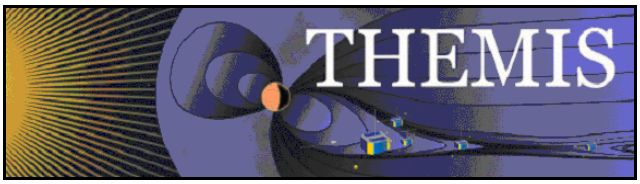
4. TDAS Branches

4.1 Introduction

The TDAS release software currently has three different release branches.: **ssl_general**, **themis**, **external**. A fourth branch **erg** contains the ERG TDAS plugin. An additional branch **QA_testing**, is part of TDAS development, but not of software releases. Each of these branches serves a different purpose and has different dependencies. It is important that new software be placed in the correct branch.

4.2 ssl_general

This contains general purpose library routines. These include routines for plotting, analysis, coordinate transformation, data structures, and programming tasks. Any routines in this branch should depend only upon other routines in this branch and on built in IDL routines. This branch contains many historical routines. Great care must be taken to ensure backwards compatibility and changes to existing routines should only be made if absolutely necessary. Note: Dependency rules are not enforced for cribs.



4.3 themis

This branch contains routines for the THEMIS mission. These include load routines, calibration routines, THEMIS specific analytical transformations, THEMIS specific coordinate transformations, routines to generate THEMIS summary plots, and the THEMIS GUI. This branch can depend upon routines in the ssl_general branch or the external branch.

4.4 external

This branch contains packages external to the TDAS software, and any routines developed to interface with those packages. This includes the IDL_GEOPACK interface to the Tsyganenko models and external developers' routines. The developers' routines directory includes additions to the THEMIS software that were not developed by THEMIS programmers, but which are nonetheless useful enough to merit release with the package. The external branch may have dependencies to routines in the ssl_general branch, but not to the themis branch. Note: Dependency rules are not enforced for cribs.

4.5 QA_testing

This branch contains routines and suites to perform quality assurance testing before a release of the TDAS software. These include automated test scripts, human run test suites, and data for regression comparisons. Dependency restrictions are not enforced for the QA_testing branch. Dependencies are formed as needed to verify the correctness of release software.
See section 5.4 below for further information on QA.

5. Development guidelines

5.1 Code

5.1.1 Introduction

To maintain support for IDL 8.1 and after, it is important that all developers maintain style that supports forward and backwards compatibility.

5.1.2 All developers

5.1.2.1 Negative Array Indexing

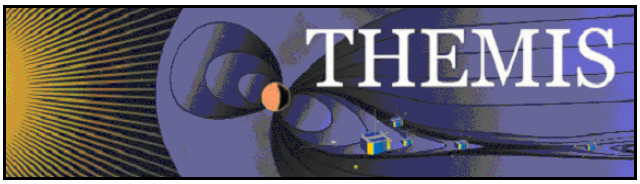
The negative array indexing feature in IDL 8.0 and after creates problems with version compatibility and error detection. To prevent errors due to negative array indexing, it is recommended that all developers not use the -1 return value from methods as a signal for an empty set. Because -1 no longer returns an error when used to index in newer version of IDL, it can create errors that are difficult to detect and silently produce incorrect results.

Avoid this:

```
index=where(array eq value)
if(index[0] eq -1) then call_function()
```

Instead use:

```
index=where(array eq value,c)
if(c eq 0) then call_function()
```



5.1.2.2 HEAP_GC

New memory management routines were introduced in IDL 8.0. If heap_gc is called in later versions of IDL 8.0 it can be harmful. Corrupting memory and causing crashes. Thus, if code is written that requires automatic garbage collection (code that would use heap_gc in earlier versions of IDL or code that calls ptr_new without ptr_free in newer versions of IDL), calls to heap_gc should be added as follows:

```
if double(!version.release) lt 8.0d then heap_gc
```

5.1.3 IDL 8.1+ developers only.

TDAS does not support new features introduced in IDL 8.0 and later. Thus no TDAS code can use these features. This includes !N ULL, list & hash, negative array indexes, foreach, & object operator overloading.

5.2 Documentation

5.2.1 Introduction

Nightly builds of TDAS automatically build HTML documentation for routines if documentation conventions are followed. It is also of the utmost importance that documentation be detailed and complete so that users can understand how to correctly use TDAS, to make sure that Scientists can trust the TDAS calculations, and to guarantee ease of code maintenance for future developers. TDAS code is documented in three ways: detailed routine header descriptions, clear code style (see Section 5.3 below), and cribs.

5.2.2 Routine Header Guidelines

a. All routine headers should go at the very beginning of the file and describe the main routine (routine from which the file takes its name). Helper functions with sub-headers should follow, with the main routine being the last function/procedure listed in the file.

b. All routine headers should start with the line ";+" and end with the line ";-". All lines between the first and last line should begin with a semicolon(";"). This requirement is necessary to build the html documentation.

c. The four lines prior to the last line of the header should be:

```
; $LastChangedBy: $
; $LastChangedDate: $
; $LastChangedRevision: $
; $URL: $
```

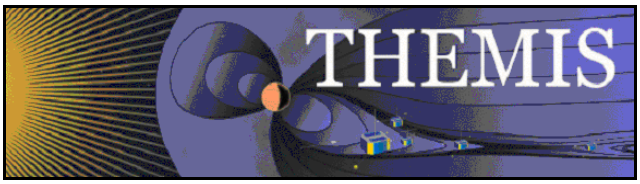
These fields will be filled in automatically by SVN.

d. All headers should contain the procedure name and a description. Sections for inputs, outputs, keywords, notes, usage, examples, references (or see also:) should be added at the discretion of the developer, with the goal of being as thorough as possible.

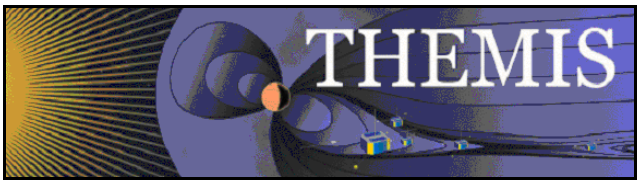
5.2.2.1 Routine Header Example

From ssl_general/mini/calc.pro

```
;+
; Procedure: calc
;
; Purpose:  This routine takes a string as input and interprets the string as a mini-language.
```




```
;    This language can manipulate normal idl variables and tplot variables.  The idl variables
;    that can be modified are non-system variables in the scope in which the procedure is called.
;
; Inputs: s : A string that will be interpreted as the mini language
;
; Keywords: error:
;              If an error occurs during processing this keyword will return a struct
;              that contains information about the error.  If error is set at time of input
;              this routine will set it to undefined, before it interprets 's' to prevent internal
errors.
;
;           function_list: return an array of strings that lists of the names/syntax of
;                          available functions in the mini_language.  Return without processing 's'
;                          if an argument in the returned name is in brackets it is optional
;                          for example:  min(x[,dim])
;           operator_list:  return an array of strings that lists of the names of available
;                           operators in the mini_language. Return without processing 's'
;
;           verbose:  set this keyword if you want the routine to print errors to screen rather
;                     than only return them via error
;
;           gui_data_obj: If 'calc' is being used inside the gui, then the loaded_data object will
;                         be passed in through this keyword.  NOTE: end users should not even set
;                         this argument.
;
; Outputs: none, but it will modify various variables in your environment
;
; Examples:
;    calc,'a = 5'
;    calc,'"pos_re" = "tha_state_pos"/6374'
;    calc,'a += 7'
;    calc,'"tvar" = "tvar" + var'
;    calc,'"tvar" = ln("tvar")'
;    calc,'"tvar" = total("tvar"+3,2)'
;    calc,'"tvar" = -a + 5.43e-7 ^ ("thb_fgs_dsl_x" / total("thb_fgs_dsl_x"))
;    calc,operator_list=o,function_list=f
;
; Notes:
;    1. The language generally uses a fairly straightforward computational syntax.  The main
;       difference from idl is that quoted strings are treated as tplot variables in this language
;    2. A full specification of language syntax in backus-naur form can be found
;       in the file bnf_formal.txt, the programmatic specification of this syntax
;       can be found in productions.pro
;    3. The language is parsed using an slr parser.  The tables required to do this parsing
;       are generated and stored ahead of time in the file grammar.sav and parse_tables.sav
;    4. The routines that describe the evaluation rules for the language can be found in the file
;       mini_routines.pro
;    5. If you want to modify the way the language works you'll probably need to modify
productions.pro,
;       regenerate the slr parse tables using save_calc_tables and modify/add routines to
mini_routines.pro
;    6. Arrays must have the same dimensions to be combines, and tplot variables must also have
the same times.
;    7. Procedures: min,max,mean,median,count,total  all take a second argument that allow you to
select the
;       dimension over which the operation is performed
;
; See Also:
;   All routines in the ssl_general/mini directory
;   The techniques used for this interpreter are based on two books:
;
;   1. Compilers:Principles,Techniques,and Tools by Aho,Sethi,& Ullman 1988 (esp. Ch3)
;
```



```
;   2. Structure & Interpretation of Computer Programs by Abelson & Sussman 1996 (esp. Ch4)
;
;   If you want to understand/modify this program it may help to use these books as
;   a reference.
;
;   Also see:  thm_crib_calc.pro for examples of usage
;
; ToDo: 1. Implement Constants
;       2. Implement 0 argument functions
;       3. Implement keywords for functions
;       4. Implement procedures
;       5. Implement control statements
;       6. Optional auto-interpolation
;
; $LastChangedBy: pcruce $
; $LastChangedDate: 2009-07-13 09:43:23 -0700 (Mon, 13 Jul 2009) $
; $LastChangedRevision: 6418 $
; $URL: svn+ssh://thmsvn@ambrosia.ssl.berkeley.edu/repos/ssl_general/trunk/mini/calc.pro $
;-
```

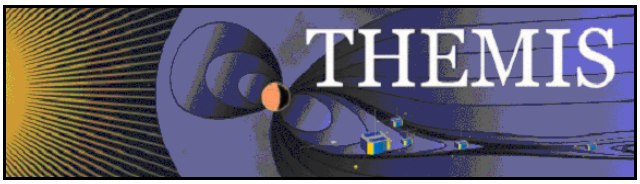
5.2.3 Crib Guidelines:

Cribs should be provided for a selection of useful or high level routines. They are intended to provide the user with an introduction to a routine, demonstrate common usage and pitfalls, and provide a base of code for users to start building from. Documentation and comments should be detailed and several different usage examples should be provided.
Cribs should ideally be provided as IDL programs that the user can either compile and run from the command line, or copy and paste the lines of interest from.

5.2.3.1 Crib example

From themis/examples/thm_crib_calc.pro

```
;+
;procedure:  thm_crib_calc.pro
;
;purpose: demonstrate how to use the mini-language program 'calc'
;
;usage:
; .run thm_crib_fgm
;
;
; $LastChangedBy: pcruce $
; $LastChangedDate: 2008-09-15 17:05:42 -0700 (Mon, 15 Sep 2008) $
; $LastChangedRevision: 3501 $
; $URL:
svn+ssh://thmsvn@ambrosia.ssl.berkeley.edu/repos/thmsoc/trunk/idl/themis/examples/thm_crib_calc.pr
o $
;
;-
;the mini-language is a language written in IDL that can be run on the
;idl virtual machine.  It follows most of the syntactical rules of IDL
;with the exception that tplot variables are treated as first-class
;data types in the mini-language.  They are denoted with "quotes" the
;same way that you would denote strings in normal IDL.
;All statements in the mini-language must be assignment statements
;(ie variable = expression)


;Example 1 var = number

;sets a equal to 5
```



```
calc,'a = 5'

stop

;Example 2 binary operation.

;This performs multiplication on "b", you can use any of the idl
;operators with calc
b = 7
calc,'seven_pi = b * 3.14159'

stop
```

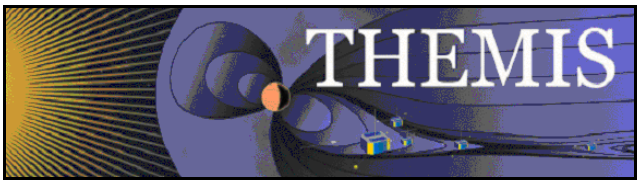
⋮
Lines omitted
⋮

```
;example 14 exp

;exp also takes a base as an optional argument.

calc,'ex = exp(2)'  ; ex = e^2

calc,'ex = exp(6,2)' ; ex = 2^6 = 64.0

calc,'ex = exp(log(64,2),2)' ; ex = 2^log_2(64) = 64

stop

end
```

5.3 Style Guidelines

Code should be written in a manner such that it is both easy for users to understand its function, and easy for future TDAS developers to develop and maintain.

a. TDAS routines follow a broad naming convention of the form ‘mission’_‘action’_‘datatype’, for example: thm_load_gmag (where ‘thm’ indicates the THEMIS mission). Routine names should clearly indicate the function of the routine.
b. In-line comments are left to the discretion of the developer. Sufficient detail should be provided so that someone who is unfamiliar with the code and has average familiarity with IDL can understand the goals, methods, and function of a routine.
c. Developers should select variable names that use consistent capitalization and underscoring across all names in a routine. Naming conventions should change only to indicate differences in type.
d. Variable names should be descriptive, not single letters or nonsense words. Good: (energy_min, mag_var_name, re_in_km,scpot_to_dens) Bad: (e,input1,foo,fx)
e. Developers should avoid the use of large (time intensive) for loops, working instead with the IDL array functions wherever possible (e.g. reform, rebin, transpose, value_locate etc).
f. It is often better to divide a long routine into a number of smaller sub-functions. This avoids code repetition, makes it easier for future users to understand the code, and for future developers to maintain the code. Note that the main routine should always be the last in the file.
g. If writing a load routine, it is important to remember to set data defaults like units, coordinate systems, and plotting settings. For colors we recommend [2,4,6] for 3-vector quantities, which corresponds to blue, green, & red.



5.4 Error handling

TDAS attempts to follow consistent error handling practices. All routines should gracefully handle error situations and give useful information to the user about the error that has occurred. Situations where a routine simply crashes should be avoided, even if the user is attempting to use the routine incorrectly.
Routines should check validity of values passed in as keywords (eg. datatypes, site names). If a user enters an incorrect value (or simply misspells something), the routine should return useful information about correct keyword values. For example:

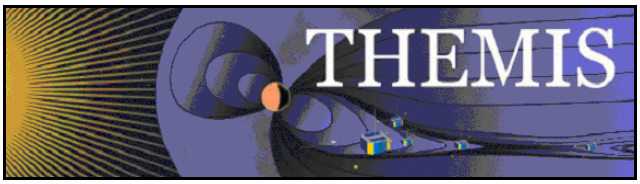
```
THEMIS> thm_load_asi, site='innu'
% Compiled module: THM_LOAD_ASI.
Level = ['l1']
% THM_CHECK_VALID_NAME: Input: innu is not valid.
% THM_CHECK_VALID_NAME: Input must be one or more of the following strings:
all atha chbg ekat fsmi fsim fykn gako gbay gill inuv kapu kian kuuj mcgr pgeo pina rank snkq tpas
whit yknf nrsq snap talo
```

In general the TDAS GUI uses two levels of error handling. Dialog messages are issued if the message is of high importance, a message is printed to the status bar and history window for less important, informational, messages. Any GUI fields that allow user input (for example, text fields, date fields) should check the validity of the user input and issue a warning message to the user if the input is invalid.

5.5 QA Scripts

Test scripts enable complete and consistent testing of all TDAS functionality prior to software release. QA scripts should be supplied that thoroughly test all functionality of routines, including error handling. These can be either automated testscripts (IDL programs) or, where necessary (e.g. testing user interaction with a GUI), detailed instructions for a human tester. See examples below, and in the QA_testing branch of TDAS.
Command line tests should have clearly defined and documented pass/fail criteria and should check for pass/fail automatically wherever possible. In cases where automated checking of test results is not possible, for example generating a plot, the script output should include a detailed description of the desired outcome or, for example, refer to an archived reference plot. Many automated testsuites follow a basic outline:

```
thm_init
t_num = -1
init_tests

; other general preamble

t_name='Name of test'

catch,err

if err eq 0 then begin

  ; test content

endif

catch,/cancel

handle_error,err,t_name,++t_num

;repeat for all tests

end_tests

end
```

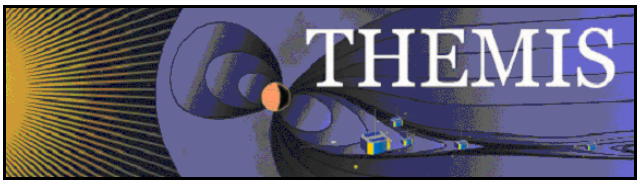


Testscripts for GUI functions should be written in such a way that minimizes the amount of setup required by the tester prior to performing the tests. If a group of tests use the same setup (eg. same data to load) consider including a step in the test to "Save THEMIS document" so that the tester can reopen the saved document at the start of each test and not need to manually reproduce the loading steps.

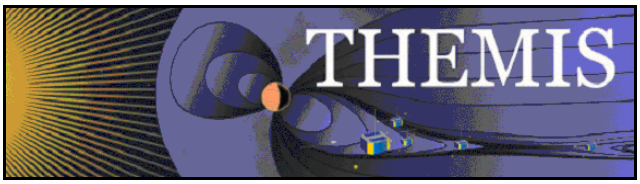
5.5.1 Command line testsuite example

From QA_testing/thm_gmag_load_cltestsuite.pro

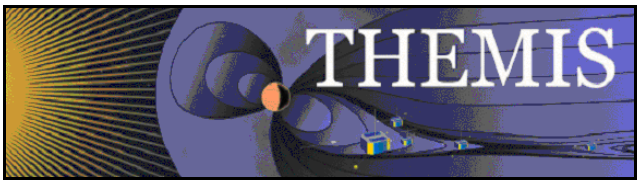
```
;0 basic
; thm_load_gmag
;
;1 single site
;
; thm_load_gmag,site='inuv'
;
;2 multi sites string
;
; thm_load_gmag,site='inuv chbg'
;
;3 multi sites array
;
; thm_load_gmag,site=['inuv','chbg']
;
;4 caps site
;
; thm_load_gmag,site='INUV'
;
;5 load all sites
;
; thm_load_gmag,site='all'
;
;6 load * sites
;
; thm_load_gmag,site='*'
;
;7 level 1 numerical
;
; thm_load_gmag, level=1
;
;8 level 1 string
;
; thm_load_gmag,level='l1'
;
;9 level 1 caps string
;
; thm_load_gmag,level='L1'
;
;10 single datatype
;
; thm_load_gmag,datatype='ask'
;
;11 caps datatype
;
; thm_load_gmag,datatype='ASK'
;
;12 * datatype
;
; thm_load_gmag,datatype='*'
;
;13 valid_names
;
```




```
;thm_load_gmag,/valid_names,probe=testp,level=testl,datatype=testd
;
;14 verbose
;
;thm_load_gmag,site='fykn',/verbose
;
;15 /downloadonly
;
;thm_load_gmag,site='fykn',/downloadonly
;
;16 /no_download
; thm_load_gmag, /no_download
;
;17 relpathnames_all
;
;thm_load_gmag,relpathnames_all=relpathnames_all
;
;18 suffix
;
;thm_load_gmag,site='fykn kian',suffix='_test'

;19 test thm_sites keyword
;
;thm_load_gmag,/thm_sites
;
;20 test dtu_sites keyword
;
;thm_load_gmag,/dtu_sites
;
;21 test tgo_sites keyword
;
;thm_load_gmag,/tgo_sites
;
;22 test ua_sites keyword
;
;thm_load_gmag,/ua_sites
;
;23 test maccs_sites keyword
;
;thm_load_gmag,/maccs_sites

init_tests

;prepare to run tests
t_num = 0

thm_init

;set to somewhere appropriate, if needed
;!themis.local_data_dir = '/disks/themisdata/'
timespan,'2008-02-22',1,/hour

del_data,'*'

;0 basic
; thm_load_gmag
;


;1 Basic Test
; thm_load_asi

t_name='basic'
```



```
catch,err

if err eq 0 then begin

thm_load_gmag

;just spot checking cause there are a lot of data types
print_tvar_info,'thg_mag_kian'

if ~data_exists('thg_mag_kapu thg_mag_mcgr thg_mag_chbg thg_mag_gill','2008-02-22','2008-02-23') $
  then message,'invalid load'

endif

catch,/cancel

handle_error,err,t_name,++t_num

del_data,'*'

;1 single site
;
; thm_load_gmag,site='inuv'
;

t_name='single site'

catch,err

if err eq 0 then begin

thm_load_gmag,site='inuv'

;just spot checking cause there are a lot of data types
print_tvar_info,'thg_mag_inuv'

if ~data_exists('thg_mag_inuv','2008-02-22','2008-02-23') $
   then message,'invalid load'

endif

catch,/cancel

handle_error,err,t_name,++t_num

del_data,'*'


;2 multi sites string
;
; thm_load_gmag,site='inuv chbg'
;

t_name='multi site string'

catch,err

if err eq 0 then begin

thm_load_gmag,site='inuv chbg'

;just spot checking cause there are a lot of data types
```



```
print_tvar_info,'thg_mag_chbg'

if ~data_exists('thg_mag_inuv thg_mag_chbg','2008-02-22','2008-02-23') $
   then message,'invalid load'

endif

catch,/cancel

handle_error,err,t_name,++t_num

del_data,'*'
```

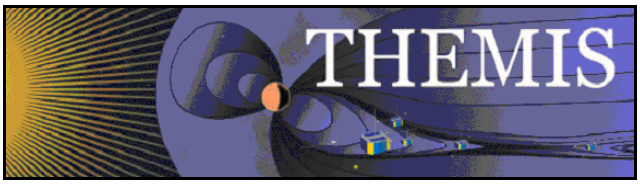
⋮
Lines omitted
⋮

```
;
;23 test maccs_sites keyword
;
;thm_load_gmag,/maccs_sites
t_name='test maccs_sites'

del_data,'*'
timespan,'2010-06-06'

catch,err

if err eq 0 then begin

thm_load_gmag, /maccs_sites

;just spot checking cause there are a lot of data types
print_tvar_info,'thg_mag_nain'

if ~data_exists('thg_mag_nain','2010-06-06','2010-06-07') $
    then message,'data load failed'
endif

catch,/cancel
handle_error,err,t_name,++t_num

del_data,'*'
end_tests

end
```

5.5.2 GUI testsuite example

GUI testscripts (in particular) should include tests of unexpected user behavior, such as entering non-numeric characters in a numeric only text field, invalid dates in a date field, or attempting to load data without selecting a datatype.
From QA_testing/thm_gui_testsuites/instruments/thm_gui_gmag.txt

```
THEMIS GUI TEST SUITE
Extended Version for GMAG Data

SUMMARY:   The purpose of this test is to run GMAG data through the system using all features and
functions possible.

TEST SETUP:
```




```
Step 1:  Run the themis gui
a)  .full_reset_session
b)      thm_gui

LOAD DATA:
Step 2:  Load data GMAG data
a)      Select the Load THEMIS Data option under the file pull down menu
b)      Use the instrument type droplist and select GMAG
c)      Enter Time Ranges
a.      Start Time of 2009-01-03/00:00:00.00
b.      Stop Time of 2009-01-04/00:00:00.00
d)      In the stations list box choose inuv, kian, snap, tpas, and whit
e)      And in level 2 select gmag
f)      Check that the correct types are displaying in the message bar at the bottom of the page
g)      Click on the top arrow (pointing towards the Data Loaded List Box)
h)      You should see GMAG data
i)      Select Clear All under the Data Loaded List Box (should empty the list)
j)      Repeat Step 2a-h


PLOT DATA:
Step 3:  Plotting inuv data
a)      Select Plot/Layout under the Graph pull down menu
b)      Click the Add button on the far right side of the window in the panels section (this will
add a new panel)
c)      From the data box, select thg_mag_inuv and click the line button next to add a line plot.
d)      Repeat 3a-b using kian and snap data.
e)      Click OK
f)      Check that there are 3 panels displayed with the correct labels/annotations
Step 4:  Add another page with GMAG plots
a)      Under the pages pull down menu, select New page
b)      Select Plot/Layout under the Graph pull down menu
c)      Repeat Steps 3a-d using snap x/y/z data, plot all 5 stations  (create 5 panels)
d)      Click apply
e)      Close page
```

⋮
Lines omitted
⋮

```
DONE!!!
```