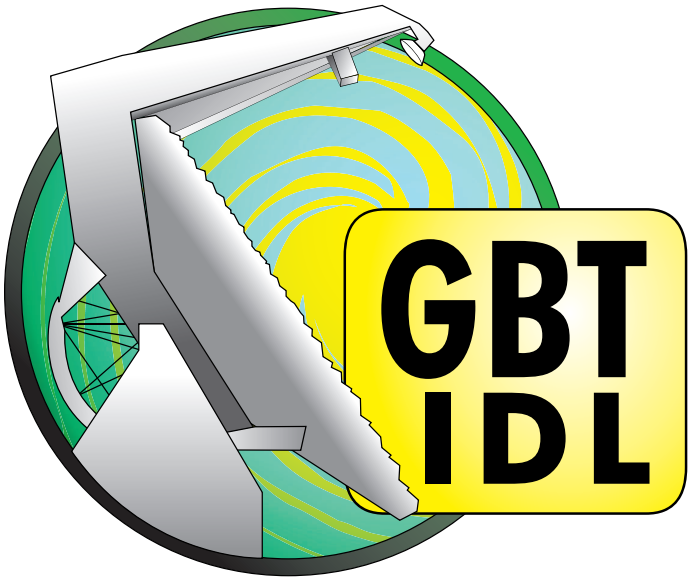
GBTIDL User's Guide

Kristen Thomas, Jim Braatz, and Bob Garwood

April 30, 2012



Contents

1 Introduction

1.1 How To Use This Document

This User's Guide is one part of the documentation package for GBTIDL. It is recommended that new users read chapters 1 through 10 sequentially. The appendices provide supplemental material. In addition to this document, there is also an online User Reference, a one-page Quick Reference Guide, several examples and FAQs, and detailed descriptions of internal data structures. All GBTIDL documentation can be accessed online from http://gbtidl.nrao.edu.

1.2 What is GBTIDL?

GBTIDL is an interactive package for the reduction and analysis of spectral line data taken with the Robert C. Byrd Green Bank Telescope (GBT). GBTIDL is written entirely in IDL. There is limited support in GBTIDL for GBT continuum data, but it is mainly intended for spectral line data from the spectrometer or spectral processor.

1.3 Main Features of GBTIDL

- **GBTIDL is easy and effective** The GBTIDL package consists of a set of straightforward, yet flexible, calibration, averaging, and analysis procedures modeled after the UniPOPs and CLASS data reduction philosophies.
- **Plotter** GBTIDL features a customized plotter with many built-in visualization features.
- **Support for advanced users** GBTIDL has Data I/O and toolbox functionality that can be used for more advanced tasks.
- **Data structures** GBTIDL makes use of data structures for storing spectra along with their headers.
- **Online feature** GBTIDL can be run in *online mode* while observing with the GBT to give users rapid access to the most recent data.

1.4 Where can I run GBTIDL?

GBTIDL is installed on the Linux computing systems at NRAO-Green Bank and NRAO-Charlottesville. It is also available for installation at other sites. An IDL license and installation with IDL 6.0 or later are required to run GBTIDL. GBTIDL has been tested on both Linux and Apple Mac installations. If you do not have an IDL license, it is possible to connect to a Green Bank computer from your remote site or sign up for time on a machine in Charlottesville that is dedicated to supporting remote users. Contact Jim Braatz or Bob Garwood for further information. In Green Bank and Charlottesville, GBTIDL is installed in */home/gbtidl/release/gbtidl*. The NRAO developed IDL code is found starting in the *pro* subdirectory.

1.5 Obtaining GBTIDL

GBTIDL is available as a tar file from the GBTIDL home page at
http://gbtidl.nrao.edu. Simply click on the **Download GBTIDL Now** link and follow the instructions.

1.6 User Documentation

The following documents are available from links on the http://gbtidl.nrao.edu homepage:

- The GBTIDL Quick Reference Guide gives a topical summary of the IDL procedures and functions.
- The User Reference provides a list of GBTIDL procedures with detailed descriptions of parameters and usage examples.
- The Contributed Code Reference describes user contributed procedures.
- Calibration of GBT Spectral Line Data in GBTIDL describes how you can optimize calibration of your data.

Other documents that may be useful to GBTIDL users include:

- The text *Practical IDL Programming* by Liam E. Gumley (2001) is a useful resource for beginning and expert IDL users.
- Using IDL to Manipulate and Visualize Scientific Data by Bill Davis (http://nstx.pppl.gov/nstx/Software/IDL/idl_intro.html)
- IDL Tutorials by Carl Heiles and Tim Robishaw (http://astro.berkeley.edu/~heiles/handouts/handouts_idl.html)

2 Quick Start Example of GBTIDL

This section shows the look and feel of the command line user interface in GBTIDL. The commands will be described later in this document.

```
% gbtidl                          ; Start GBTIDL from the unix prompt

GBTIDL -> filein                  ; Specify an input file using the file selection GUI
GBTIDL -> summary                 ; Give a summary of the scans in the opened data file
GBTIDL -> getfs, 9                ; Retrieve, calibrate, and plot a frequency switched
                                  ;  spectrum (other observing modes use different commands)

GBTIDL -> setregion               ; Set region used to determine baseline
GBTIDL -> nfit,2                  ; Specify that a 2nd order baseline will be used
GBTIDL -> baseline                ; Fit and subtract a baseline

GBTIDL -> fitgauss                ; Fit a Gaussian profile to the plotted spectrum.  The mouse
                                  ;  is used to set initial guesses

GBTIDL -> stats                   ; For the displayed spectrum, show statistics such as
                                  ; the RMS, maximum and minimum x and y values
GBTIDL -> print_ps                ; Write the current spectrum to a postscript file

GBTIDL -> fileout, 'mydata.fits' ; Specify the output file name for saved data
GBTIDL -> keep                    ; Save the spectrum currently displayed

GBTIDL -> exit                    ; Exit GBTIDL
```

3 Getting Started

This section describes how to begin a GBTIDL session and what to do if you need help understanding a command.

3.1 Starting GBTIDL

To start GBTIDL simply type the following command at the unix prompt from any NRAO computer running Linux in Green Bank or Charlottesville.

```
% gbtidl
```

3.2 Getting Help with Commands

In addition to the GBTIDL User Reference, you can get help within GBTIDL via the **usage** command. For example, to get help with the **show** command:

- `usage, 'show'` to get a list of the optional arguments.

```
GBTIDL -> usage, 'show'

Usage: show[, dc ] [, color=color ] [, /defaultx ] [, /smallheader ] [, /noheader ]
```

- `usage, 'show', /verbose` to get a complete description of what the command does, what keywords can be used, examples, and the path for the source code.
- `usage, 'show', /source` to see the source code behind the command.

4 Accessing Data Files

GBTIDL reads SDFITS files generated by the **sdfits** filler program run in Green Bank. During an observation the data are automatically filled into the SDFITS format in real time and these data can be accessed in *online* mode. After an observing session, data are accessed in *offline* mode.

4.1 Working Online

In Green Bank, recent raw (unfilled) data can be found in `/home/gbtdata`. As scans are completed, SDFITS files and their corresponding index files and an empty flag file are automatically generated in the directory `/home/sdfits`. Files are grouped into subdirectories by project name. Note that SDFITS files are only produced for the Spectrometer, Spectral Processor, and Zpectrometer. No SDFITS files are produced automatically for the DCR..

To view data being filled into the online directory, simply issue the command **online**. The project name and location of the online data file do not need to be specified. This command also sets GBTIDL into online mode so subsequent GBTIDL commands automatically detect new data as they are filled. There is no need to create SDFITS and index files manually, nor any need to issue special commands to pick up the latest scans. For example:

```
GBTIDL -> online                        ; Connects to most recent project's online SDFITS file

Connecting to file: /home/sdfits/project/project.raw.acs.fits
File has not been updated in 2.35 minutes.

GBTIDL -> getfs, 10                     ; get some recent data, while scan 11 is integrating
GBTIDL -> getfs, 11                     ; when scan 11 is finished, its data are available with
                                        ; no additional filein command necessary
```

If problems are encountered with the online system, contact the on-duty telescope operator.

Note that all data in `/home/sdfits` is *read only* except for the flag files, which need to be writable so that GBTIDL users can flag and unflag the online data.

GBTIDL will remain in online mode until an offline data set is specified for input using the **offline**, **filein** or **dirin** command.

4.2 Accessing SDFITS Data After an Observing Run

Data created by the online SDFITS filler will remain in the `/home/sdfits` directory for approximately 3 months before it is automatically deleted. You may wish to preserve your SDFITS files by copying them from `/home/sdfits/project` to your own account (where "project" here is the name of the project that you wish to preserve).

To connect to data in the `/home/sdfits` directory after the project has finished observing, you can either use the **filein** command (described in section 4.3) or the **offline** command:

```
GBTIDL -> offline, 'AGBT06C_028_05'
```

4.3 Creating SDFITS Files

After about 3 months time, the SDFITS data in `/home/sdfits` are deleted and the "unfilled" data in `/home/gbtdata` are moved to `/home/archive`. You will need to create new SDFITS files in your account if you haven't copied the originals or if the default filling method is not what you want. If you are storing large data files in Green Bank it is preferred that you use scratch space, such as the `/home/scratch` area. This area is meant for temporary storage of large data files.

To create SDFITS files from data in `/home/gbtdata` or `/home/archive`, use the **sdfits** program, which runs from the unix prompt, only in Green Bank. Help is available for the **sdfits** program as follows:

```
% sdfits -help
```

In addition to the online help, the **sdfits** program has full documentation at http://safe.nrao.edu/wiki/bin/view/GB/Data/Sdfits including development status, known issues, request list, and usage examples.

The **sdfits** program offers three levels of calibration, identified by the -mode switch as "raw" (default), "cal", or "avg". Users of GBTIDL will generally want to use the default of *no calibration* (-mode=raw) and use GBTIDL routines to do the calibration, instead. With -mode=raw, the SDFITS file will contain one row of data for each data phase. That is, sig_calon, sig_caloff, ref_calon, and ref_caloff phases are all stored individually in the SDFITS file. The other modes (cal and avg) are not recommended since they use un-maintained, older calibration code. In the future these will use the same calibration code used by GBTIDL.

The **sdfits** program writes data into the current directory by default, so it is best to change into a directory in which you can write large files before running **sdfits**. In Green Bank, you have a quota on your home directory so it is best not to fill large datasets into your home area.

Typical use looks like this:

```
% cd /home/scratch/[username]
% sdfits -scans=1:100 /home/gbtdata/AGBT01A_001_01
```

The resulting SDFITS file, assuming it contains spectrometer (ACS) scans, is called AGBT01A_001_01.raw.acs.fits. If the specified scan range also includes data from the DCR or Spectral Processor, then a separate output file will be created for each backend.

4.4 Accessing SDFITS Files

The **filein** command is used to identify an SDFITS file as the source of input. The name of the SDFITS file can be supplied as a parameter. If the file is in the directory from which GBTIDL was started, the full path is not required. For example:

```
GBTIDL -> filein,'mydata.fits'
```

However, if GBTIDL was not started in the directory in which the file is stored, you must include the path. For example,

```
GBTIDL -> filein,'/users/aeinstein/mydata.fits'
```

If you omit the filename and simply type `filein`, you can select the input file using the file selection GUI.

GBTIDL associates an index file with each SDFITS file. If the file *mydata.fits* does not already have an up-to-date index file, it will be created when the **filein** procedure is run. The index file, in this case, would be called *mydata.index* and would reside in the same directory as the SDFITS file. The index file is simply an ASCII listing containing information about each row in the SDFITS file. It is used in the GBTIDL I/O system to speed data retrieval and enable searching the data.

4.5 Accessing Multiple SDFITS Files Simultaneously

Accessing Multiple SDFITS Files Simultaneously It is easy to work with data from multiple SDFITS files at the same time in GBTIDL, provided the data are stored in a single directory. Rather than using **filein**, simply use **dirin** to identify the directory and all the SDFITS data in that directory will be available to the Data I/O commands. A single index file (called "dir.index") is created and there will be a unique index for each row of data throughout the directory. The **dirin** command takes one parameter, the name of the directory:

```
dirin,'/users/aeinstein'
```

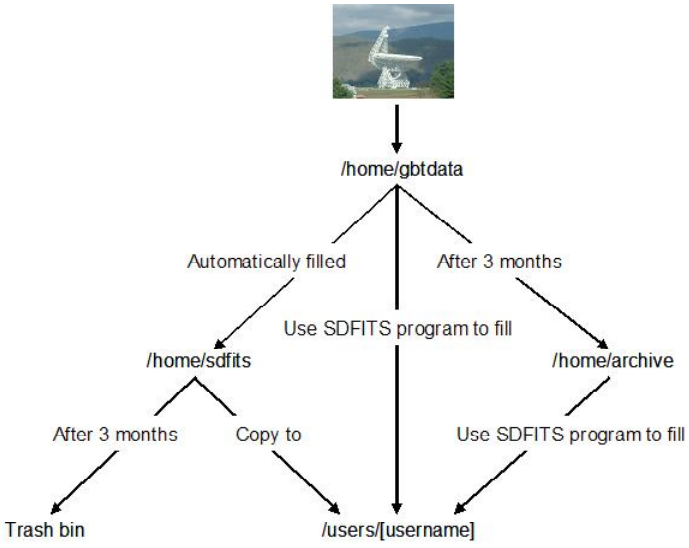
You can also simply type **dirin** and select the desired directory from the file selection GUI. Since scan numbers may repeat from one observing session to another (even within the same project) it often is simpler to work with one SDFITS file at a time. However, GBTIDL does provide ways to work with multiple occurances of the same scan number. For example, data access and calibration commands such as **getfs** have a parameter called *instance* that allows you to select which instance of a given scan number to acquire.

4.6 Summary of the Location of Data

1. Recent raw data can be found in `/home/gbtdata`.
   - These files need to be converted to SDFITS before they can be used in GBTIDL.
   - After approximately 3 months, these files are moved to `/home/archive`.
2. Data are archived in `/home/archive` for long term storage.
   - These files need to be converted to SDFITS before they can be used in GBTIDL.
3. While observing, SDFITS files are produced automatically in `/home/sdfits/project`.

- The **online** and **offline** commands in GBTIDL know about and can use these files.
- Files in `/home/sdfits` are removed as needed to make room for newer data. Sdfits files will remain available for at least 3 months and often longer. They are not archived since they can be recreated from the data in `/home/gbtdata` or `/home/archive`.
- The data in `/home/sdfits/project` was filled using defaults that may not be right for you. You may want to create SDFITS files using your own choice of **sdfits** settings.
- "project" in /home/sdfits/project is the name of the project being observed (e.g. AGBT01A_001_01).

4. SDFITS files in user accounts are for longer term storage.
   - The **sdfits** program creates SDFITS files in your personal acoount.
   - The **filein** and **dirin** commands in GBTIDL work on files in your account.



4.7 Listing the Contents of Data Files

After a file is loaded into GBTIDL using one of the methods described above, there are several ways to summarize the contents of that file.

- **summary:** This procedure gives a summary of the scans in an input data file. You can either print the summary to the screen or to a file. For example, to print to the screen, the command and its result would be:

```
GBTIDL -> summary
  Scan           Source      Vel    Proc Seq    RestF nIF nInt nFd     Az    El
-------------------------------------------------------------------------------
    79             W3OH    -44.0   Track   0    1.667   2    6   1  379.2  16.1
    80             W3OH    -44.0   Track   0    1.667   2    6   1  379.4  16.2
    81             W3OH    -44.0   Track   0    1.667   2    6   1  379.5  16.3
    82             W3OH    -44.0   Track   0    1.667   2    6   1  379.6  16.4
    83             W3OH    -44.0   Track   0    1.667   2    6   1  379.8  16.4
```

If you wish to save the summary to a file, use:

```
GBTIDL -> summary, 'myfile.summary'
```

- **list:** The *records* in the input data set can be summarized with the **list** command. You can choose to list all of the records or only a subset of them using optional parameters. To display a range of records, two optional parameters are needed, a beginning and ending record number. For example, the following will list the first 11 records in the input data set:

```
GBTIDL -> list,0,10
#INDEX           SOURCE       SCAN PROCEDURE POL IFNUM FDNUM        INT SIG CAL
     0            W3(OH)         5     Track  XX     0     0          0   F   T
     1            W3(OH)         5     Track  XX     0     0          0   T   T
     2            W3(OH)         5     Track  XX     0     0          0   F   F
     3            W3(OH)         5     Track  XX     0     0          0   T   F
     4            W3(OH)         5     Track  XX     0     0          1   F   T
     5            W3(OH)         5     Track  XX     0     0          1   T   T
     6            W3(OH)         5     Track  XX     0     0          1   F   F
     7            W3(OH)         5     Track  XX     0     0          1   T   F
     8            W3(OH)         5     Track  XX     0     0          2   F   T
     9            W3(OH)         5     Track  XX     0     0          2   T   T
    10            W3(OH)         5     Track  XX     0     0          2   F   F
```

For more details, type: `usage, 'list', /verbose`

- **files:** The **files** command prints to the terminal screen the file names being used for I/O. For example, if you have loaded both continuum and spectral line data and issue the **files** command:

```
filein, '/home/line.fits'
cont
filein, '/home/continuum.fits'
files, /full                    ; print full path names

 spectral line  in : /home/line.fits
 spectral line out : /home/GBTIDL_KEEP.fits

     continuum  in : /home/continuum.fits
```

Note that GBTIDL_KEEP.fits is the default output file and it is opened automatically on startup.

5 Data Containers

5.1 Overview

To make effective use of GBTIDL, you must have an understanding of data containers and how they are used by GBTIDL procedures. The following sections give an introduction to data containers. Advanced users may wish to refer to the "Data Containers for Expert IDL Users" section in the Appendix.

A data container (DC) is an IDL data structure used by GBTIDL to store a spectrum and its associated header. A data container uses standard data types (e.g. integers, floats and strings) for header values and an IDL pointer for the data itself. There are 16 global data containers, or buffers, numbered 0 through 15 that are used like memory entries in a calculator. Buffer 0 contains the primary data container (PDC). The PDC will be discussed in more detail in a later section.

GBTIDL includes a number of calibration, averaging, and analysis procedures that work with the PDC by default. You may see some of these routines refered to as part of the "GUIDE layer" elsewhere in the documentation.

In addition to the global buffers, it is possible to define IDL variables as data containers. GBTIDL includes commands and procedures that operate on these as well, and you may see some of these routines refered to as part of the GBTIDL "toolbox."

The IDL global structure `!g` is central to the GBTIDL implementation. It contains not only the global data containers, but also a set of values which assist in data reduction. For example, the field `!g.nfit` stores the order of the polynomial for the baseline to be fit, so that value does not have to be specified each time the baseline procedure is run. The **gstatus** procedure summarizes the current contents of the `!g` structure. Many users will never need to interact directly with the `!g` structure. Appendix A lists all of the items in the `!g` structure.

A second global structure called `!gc` is used to store various constants . This structure is read-only. It's contents are listed in the following table.

Name	Type	Value	Description
!gc.light_speed	double	2.99792458D8	The speed of light in m s^{-1}
!gc.light_c	double	2.99792458D5	The speed of light in km s^{-1}
!gc.plank_h	double	6.6260755D-27	Planck constant in g cm^2 s^{-1}
!gc.newt_g	double	6.67259D-8	Newton gravitational constant in dyne cm^2 gm^{-2}
!gc.boltz_k	double	1.380658D-16	Boltzman constant in erg K^{-1}
!gc.eV2erg	double	1.60217733D-12	1 eV to ergs
!gc.AU	double	1.4960D+13	AU in cm
!gc.m_H	double	1.673534D-24	Hydrogen mass in gm
!gc.m_e	double	9.1093897D-28	Electron mass in gm
!gc.pc	double	3.0857D+18	1 parsec in cm
!gc.rad_sig	double	5.67051D-5	Radiation constant in erg cm^{-2} s^{-1} K^{-4}

For example, if you wish to use the Planck equation, $\Delta E = h\nu$, for a frequency of 1665 MHz, the following would calculate the energy:

```
GBTIDL -> print, !gc.plank_h * 1665d6         ; Print the product h * 1665 MHz to the screen
   1.1032416e-17                              ; The result
```

5.2 Data Containers and Pointers

A data container is an IDL data structure that uses standard data types for header information, but uses a pointer for the data array. The reason for using a pointer is that it allows the data container to hold a spectrum with an arbitrary number of channels. The continuum data container uses additional pointers to hold auxilliary information stored in arrays that must match the dimensionality of the data array (e.g. the time at the center of each integration and the pointing direction).

Pointers make the data container very flexible. Unfortunately, they also make it somewhat more difficult to use the data directly. GBTIDL procedures hide all of the nuances of working with pointers, so nearly all users will not need to be concerned with how to work with pointers in IDL. It is possible to copy the data in a data container to a standard IDL array, and back again into the data container using the GBTIDL function **getdata** and the command **setdata**. An example follows later in this section. Users who wish to learn more about advanced manipulation of data containers should see the 'Data Containers for Expert IDL Users' section in the Appendix.

5.3 About the Primary Data Container

The PDC is the data container that is used by default by many GBTIDL procedures to improve ease of use. So a display operation such as **show** will display the PDC unless told to do otherwise, and likewise for smoothing operations, statistics, and so on. Data access procedures copy the data from disk

into the PDC, unless told otherwise. Like all data containers, the PDC includes both the data and the associated header information. The PDC is stored in the IDL global variable `!g.s[0]` if it is a spectrum, and `!g.c[0]` if it is continuum data. In addition to the PDC, there are 15 global data containers that can be used for storage of results during data reduction sessions. These are called `!g.s[1], !g.s[2]`, ... and `!g.c[1], !g.c[2]`, ...

5.4 Examining and Changing Data Containers

You may on occassion need to change the contents of a data container. For example, you may need to set the rest frequency by hand. The change can be made using the following command:

```
!g.s[0].line_rest_frequency=1667.359d6    ; Change to 1667.359 MHz
```

Another example involves setting the y-axis label on the plotter. For more information about changing axis labels, see Appendix H.

```
!g.s[0].units='F(Jy)'     ; Set the label to 'F(Jy)'
```

To access the array containing the actual data values in a data container, use the commands **getdata** and **setdata** . For example:

```
GBTIDL -> getrec, 0                          ; get some data
GBTIDL -> mydata = getdata()                 ; copy the data array into an IDL variable
GBTIDL -> help,mydata
MYDATA          FLOAT     = Array[8192]
GBTIDL -> mydata[0:500] = 0                  ; make some changes to the IDL array
GBTIDL -> setdata,mydata                     ; insert the new array into the PDC
```

5.5 Data Container Operations

GBTIDL can be used as a calculator, operating on spectra contained in the 16 global data containers. Procedures are available to perform arithmetic operations on the global data containers, including **add**, **subtract**, **multiply**, and **divide**. These procedures take two required parameters: the indices of the buffers being operated on. They also take an optional third parameter, which identifies the buffer into which the result will be stored. If a storage buffer is not specified, the result is placed in the PDC (buffer 0), overwriting any existing spectrum there. A **copy** command copies the contents of one buffer directly into another. For example, to add two data containers, you could use the following command sequence:

```
getrec,1                ; Get some data
copy,0,10               ; Copy the PDC to DC 10
getrec,0                ; Get some other data
copy,0,11               ; Copy the PDC to DC 11
add,10,11,12            ; Put the sum of the two spectra in DC 12
show,12                 ; Show the sum
```

These operations can be useful for handling baseline subtraction. For example, you can store a baseline fit in a data container and subtract that fit from any other spectrum, as in the following example:

```
getrec, 0                   ; get spectrum A and place it into the PDC (buffer 0)
nfit, 5                     ; set the order for the polynomial in a baseline fit
bshape, modelbuffer = 10    ; fit a baseline and store it in buffer 10
getrec, 1                   ; get spectrum B
copy, 0, 5                  ; copy spectrum B into buffer 10
subtract, 5, 10, 11         ; subtract the spectrum A baseline from spectrum B
```

6 Data Retrieval and Calibration

6.1 Calibrating Data

If the **sdfits** program was run with -mode=raw as recommended, the data in the SDFITS file are uncalibrated. The observer can calibrate the data in GBTIDL. If the data were taken in one of the GBT standard observing modes supported by GBTIDL, then there is a GBTIDL command that can be used to retrieve and calibrate the data. GBT calibration can be complex. These procedures typically give results accurate to 10% - 20%. If you require higher precision, refer to the document *Calibration of Spectral Line Data in GBTIDL.*

For spectral line data, the following calibration procedures are available:

```
Frequency Switched             :  getfs
Total Power Position Switched  :  getps or getsigref
Total Power Nod                :  getnod
Beam Switched                  :  getbs
```

Refer to the User Reference or use the **usage** command in GBTIDL for details on using these commands. For example, to get information about **getfs**, use:

```
usage, 'getfs', /verbose
```

Each of the calibration procedures except **getsigref** takes one required parameter: the M&C scan number. In the case of **getfs**, the calibration and retrieval pertains to a single scan. For **getps** and **getnod**, the data comes in scan pairs and the scan parameter can be either scan in the observing procedure. For example, if an *OffOn* observation comprises scans 9 and 10, then the following commands give the same result:

```
getps,9
```

and

```
getps,10
```

The procedure **getsigref** takes two required parameters: the scan used for the "signal" and the scan used for the "reference". So **getsigref** offers some flexibility to use mismatched sig/ref pairs or data from non-standard procedures.

```
getsigref, 14, 21  ; Get and calibrate data with signal scan 14 and reference scan 21
```

Each of the calibration procedures takes optional parameters. A few of the data selection keywords are listed below:

Keywords related to data selection:

Keyword	Description	Default Value
ifnum	spectral window index	0
intnum	integration number	all integrations averaged
plnum	polarization index; 0=LL or XX, 1 = RR or YY	0
fdnum	feed number	0
sampler	sampler name; alternative to ifnum,plnum,fdnum	unused unless given explicitly

So, for example, to retrieve the polarization LL data from the second IF, third integration in scans 12-13, which make up a total power NOD observation, one could use:

```
getnod, 12, ifnum=1, plnum=0, intnum=2
```

Unlike some data processing packages, GBTIDL does not automatically average the two polarizations associated with a scan. So, you must be sure to average polarizations by hand where appropriate.

Keywords to control details of the calibration:

Keyword	Description	Default Value
smthoff	smooth the off spectrum by smthoff channels	no reference smoothing
tsys	system temperature	Tsys derived from the data
tcal	cal temperature	taken from the data header
eqweight	apply equal weighting to integrations when averaging	0 (false) - weight by Tsys
tau	zenith opacity	get_tau(freq)
ap_eff	aperture efficiency	get_ap_eff(freq)
units	units of 'Ta', 'Ta*', or 'Jy'	'Ta'

Other keywords

Keyword	Description	Default Value
quiet	suppress messages printed to the screen	False (0)
keepints	save the individual integration results to the keep file	False (0)
useflag	use all or some of the flag rules by id string (see flagging)	use all flag rules
skipflag	skip all or some of the flag rules by id string (see flagging)	do not skip any rules
instance	for multiple occurances of the same scan, choose this instance	0 (the first instance)
file	for multiple occurances of the same scan, find it in this file (relevant with dirin only)	first file
timestamp	for multiple occurances of the same scan, find the one with this timestamp	

6.2 Retrieving Individual Records

For most users, the data retrieval and calibration procedures discussed in the previous section will be sufficient. Others may need to access data in its raw, uncalibrated form. There are two commands for accessing uncalibrated data, **get** and **getrec**. The **getrec** command is also useful for retrieving calibrated data from a *keep* file (a file which contains data already calibrated in GBTIDL and stored using the **keep** command).

get: The **get** procedure can be used to retrieve individual data records from the input data file based on the scan number, feed, IF, integration, polarization, cal state, and sig/ref state of the data. If these parameters are not sufficient to uniquely identify a single row in the SDFITS file, only the first matching row is returned and a warning message is printed. The **get** procedure might be used as follows to calculate a system temperature from an uncalibrated data file:

```
get, scan=10, pol='LL', ifnum=1, fdnum=1, int=1, sig='T', cal='T'
calon = getdata()
get, scan=10, pol='LL', ifnum=1, fdnum=1, int=1, sig='T', cal='F'
caloff = getdata()
tcal = !g.s[0].mean_tcal
tsys = caloff/(calon-caloff)*tcal
print,'Mean Tsys = ',mean(tsys)
```

A complete list of parameters for the **get** procedure is given below:

Parameter	Description
index	record number
project	project ID
file	SDFITS file names (only relevant if the input data set is specified with dirin rather than filein)
timestamp	scan timestamp as YYYY_MM_DD_HH:MM_SS
extension	SDFITS extension number
row	SDFITS row number
source	source name
procedure	procedure name
procseqn	procedure sequence number
scan	M&C scan number
polarization	polarization, e.g. 'LL', 'RR', 'XX' or 'YY'
plnum	polarization index, zero-based
ifnum	IF (i.e. spectral window) index number, zero-based
feed	feed name (e.g. B1)
fdnum	feed index number, zero-based
int	Integration number
numchn	number of channels in the spectrum
sig	'T' or 'F' to identify SIG state
cal	'T' or 'F' to identify cal-on or cal-off
sampler	backend sampler name
azimuth	antenna azimuth
elevation	antenna elevation
longitude	longitude-like axis, e.g. RA
latitude	latitude-like axis, e.g. DEC
lst	LST
centfreq	center frequency in Hz
restfreq	rest frequency in Hz
velocity	source velocity in km/s
freqres	frequency resolution in Hz
freqint	frequency interval (channel spacing) in Hz
dateobs	date-time value
bandwidth	bandwidth in Hz
exposure	exposure time
tsys	system temperature
nsave	nsave index
trgtlat	latitude coordinate of source
trgtlon	longitude coordinate of source
obsid	observation ID
subref	subreflector state (subref_state); 0=moving, 1=first position, -1=second position

getrec: To retrieve an individual record, use the **getrec** procedure. This procedure takes one parameter, the record number. The record number is equivalent to the row number in the SDFITS file. Like all indices in IDL, the record number is a zero-based index. So, for example, the fifth record can be retrieved and displayed as follows:

```
getrec,4
```

6.3 Getting Scan Header Information

After data have been loaded into the PDC using one of the GBTIDL calibration procedures, **get**, or **getrec**, the **header** command can be used to show header information for that scan. For example:

```
GBTIDL -> header
--------------------------------------------------------------------------------
Proj: TREG_050627      Src   : W3OH                          Obs : Jim Braatz

Scan:        79        RADec :  02 27 04.1  +61 52 22     Fsky:   1.667696 GHz
Int :         0        Eqnx  :  2000.0                    Frst:   1.667359 GHz
Pol :        YY        V     :  -44.0        OPTI-LSR     BW   :  50.000    MHz
IF  :         0        AzEl  :  379.232      16.105       delF:   3.052    kHz
Feed:         1        Gal   :  133.948       1.064       Exp :  26.2      s
```

```
Proc:     Track        UT    : +04 10 20.0   2005-06-28   Tcal:    1.45    K
Sub :         0        LST/HA: +17 16 29.4   -9.18        Tsys:   28.38    K
--------------------------------------------------------------------------------
```

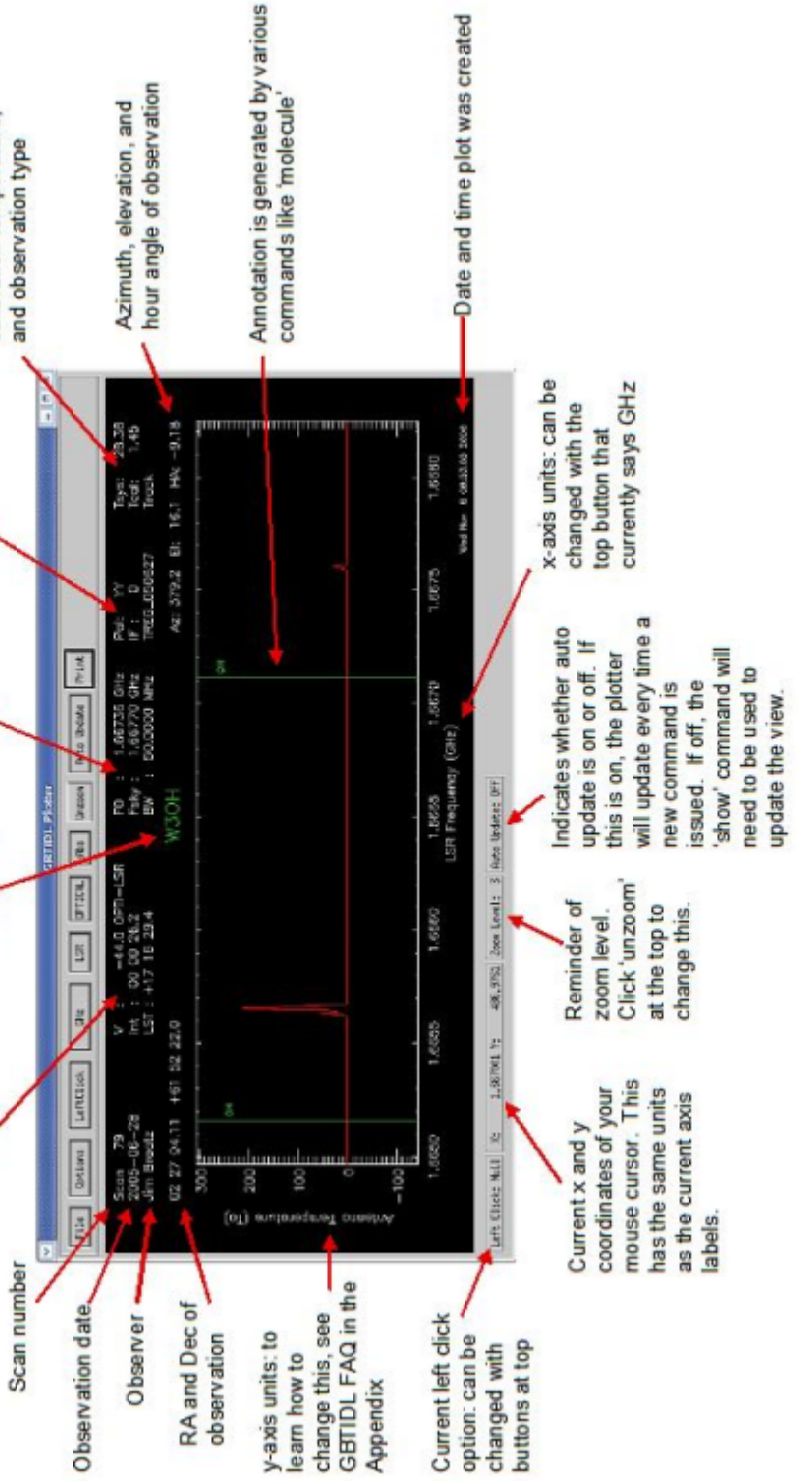
The **header** command shows information for the PDC by default, but headers for other data containers can be displayed by specifying the desired buffer index, or by specifying the IDL variable name explicitly. The following two commands are equivalent, and show the header for the data stored in buffer 2.

```
GBTIDL -> header, 2
GBTIDL -> header, !g.s[2]
```

7 The Plotter

7.1 GUI Features

When GBTIDL is first started it does not show the plotter screen, but the first time a command is issued that uses the plotter, it will appear. The figure below identifies the parts of the plotter GUI. You can manipulate the view using either command line procedures or the buttons on the plotter screen. Control buttons are positioned along the top menu bar, and status indicators are along the bottom.


Scan polarization, intermediate frequency number, and name of data file
System temperature, calibration temperature, and observation type
Azimuth, elevation, and hour angle of observation
Annotation is generated by various commands like 'molecule'
Date and time plot was created
Rest frequency, sky frequency, and bandwidth
Source name
Source velocity, integration time, and local sidereal time
Scan number
Observation date
Observer
RA and Dec of observation
y-axis units: to learn how to change this, see GBTIDL FAQ in the Appendix
Current left click option: can be changed with buttons at top
Current x and y coordinates of your mouse cursor. This has the same units as the current axis labels.
Reminder of zoom level. Click 'unzoom' at the top to change this.
Indicates whether auto update is on or off. If this is on, the plotter will update every time a new command is issued. If off, the 'show' command will need to be used to update the view.
x-axis units: can be changed with the top button that currently says GHz

Buttons for manipulating the plotter view and their equivalent commands:

- **File:** The file menu is used to print the plot or to write the data to an ASCII file. The available options are:
  - **Print...:** Print the screen (this allows you to choose your printer and print options)
  - **Write PS:** Save the plot to a postscript file
  - **Write ASCII:** Write the data to an ASCII file
  - **Exit:** Exit the plotter
- **Options:** The Options menu includes the following capabilities:
  - **Crosshair:** Toggles the cursor between a crosshair style and a pointer style. The crosshair is useful for reading x and y values off the plot.
  - **Zeroline:** Toggles a horizontal line at y=0 on the plot.
  - **Toggle Histogram:** Switches between histogram-style and connected-points style plots.
  - **Toggle Region Boxes:** Affects region boxes that were created using the **setregion** command.
  - **Clear Marks:** Clears all the markers you have made on the plotter.
  - **Clear Vertical Lines:** Clears lines you have added to the plotter.
  - **Clear Overlays:** Clears spectra overlaid on the original.
  - **Toggle Overlays:** Toggles the display of any overlays without affecting the scaling of the axes.
  - **Clear Annotations:** Clears all textual annotations on the plotter, including the results of Gaussian fits shown on the plot.
  - **Set Voffset=Vsource:** Sets the offset velocity equal to the source velocity obtained from the header.
  - **Set Voffset=0:** Sets the offset velocity to zero.
  - **Set Voffset:** Prompts the users for a new offset velocity.
- **LeftClick:** This menu lets you choose the behavior of the left mouse button. The options are:
  - **Null:** A left click does nothing.
  - **Position:** A left click will print the x and y coordinates of the cursor on the terminal screen.
  - **Marker:** A left click places a marker on the plot and displays the x and y coordinates of that marker.
  - **Vline:** A left click places a vertical line on the plot and displays the x and y coordinates of the click point.
- **X-axis units:** This button can be used to specify the desired x-axis units. The button's label is the current x-axis units. The options are:
  - **Channels, Hz, kHz, MHz, GHz, m/s**, or **km/s**
- **Reference Frame**: This button provides options for changing the velocity frame of reference. The button's label is the current velocity frame of reference. You can choose:
  - **TOPO:** Topocentric - the observed (sky) frame
  - **LSR:** Local standard of rest (kinematic)
  - **LSD:** Local standard of rest (dynamic)
  - **GEO:** Geocentric

  - **HEL:** Heliocentric
  - **BAR:** Barycentric
  - **GAL:** Galactocentric

- **Velocity Definition:** This button can be used to set the velocity definition. The button's label is the current velocity definition. Options are:
  - **Radio, Optical**, or **True** (Relativistic)
- **Abs**: This button allows you to choose whether to display the x axis in absolute units or relative to the center of the band.
- **Unzoom:** This button unzooms the plotter one step at a time. That is, if you have zoomed the plot three times successively, clicking this button once will return you to the zoom parameters applied after the second zoom. Clicking it twice more will return you to the full unzoomed scale. This button is grayed out when fully unzoomed. For more information on zooming methods, see the section on zooming below.
- **Auto Update:** This setting controls whether or not the plotter automatically responds to commands. The feature is described in the next section.
- **Print:** The print button sends the plot, as displayed in the plotter, immediately to the default printer as set in the !g.printer variable. If you want to specify a printer, use the print option under the File button.

7.2 Auto Update (Freeze/Unfreeze)

The Auto Update feature determines how the plotter responds to data processing commands. With Auto Update on, a command that changes the PDC will trigger the plotter to update with the new result. With Auto Update off, the plotter is only updated in response to a **show** command. In most cases, you will want the auto update turned on (unfreeze) so the **show** command is not required at each step. However, setting it off can be useful for faster processing of data in scripts because plotting the spectra during intermediate steps can be time consuming. From the command line, use **freeze** or **unfreeze** to turn the auto-update off or on, respectively.

For example:

```
unfreeze                        ; Turn on auto updates
getrec,1                        ; Get some data - note the plot updates
hanning                         ; The plot updates after the smooth operation
freeze                          ; Turn off auto-updates
for i=101,200 do begin & $      ; This loop will be faster
  getrec,i & $                  ;   since the plots are not updating
  accum & $
end
ave
show                            ; Now the plot updates
unfreeze                        ; back to the usual setting
```

7.3 Zooming

Zooming in on a plot can be accomplished in several ways. One is to use the middle mouse button on the plotter, clicking twice to specify the corners of the new zoom box. To unzoom, click the *Unzoom* button at the top of the plotter, or simply type **unzoom**. The *Unzoom* button takes you back to the previous zoom settings, so several clicks may be necessary to return to the full scale. However, typing

unzoom in the terminal window will bring you back to the original unzoomed spectra, no matter how many times you zoomed. If you wish to cancel a zoom after the first middle mouse click, click the right mouse button.

Zooming may also be accomplished with the **setxy** procedure. When used with no parameters, this procedure places a stretchable box on the plot and allows it to be positioned before executing the zoom. Instructions for its use are printed to the screen when the procedure is invoked. Alternatively, you can specify the desired zoom range from the command line using: `setxy, x1, x2, y1, y2`.

A third zooming method is to specify minimum and maximum x- or y-axis values using the commands **setx** or **sety**. You can then either specify the minimum and maximum x- or y-range using parameters, or omit the parameters and use the cursor to set the range. The commands **freex** and **freey** can be used to autoscale the x- or y-axis without unzooming the other axis. For example, **freey** will show the full y-range of the data without changing the current x-range.

7.4 Printing Spectra and Creating Postscript Plots

Generating postscript plots can be difficult in IDL. In GBTIDL, we have simplified the process with the **write_ps** procedure. This procedure will generate a postscript file that reproduces the plot as shown on the plotter. The postscript rendition will include overlays, show the zero line if it is turned on, show any annotations created with the **annotate** procedure, display any *markers* or *vlines* placed on the plot, and the axis ranges are accurately reproduced. However, the **write_ps** procedure cannot know about any other IDL primitives that may have been used to draw on the GBTIDL plotter, so any IDL primitive plot commands will not be reproduced.

7.5 Generating ASCII Data

The command **write_ascii** can be used to write data to an ASCII file. The command takes a single parameter, the name of the ASCII file to be generated.

The **table** command is useful for printing the x and y coordinates of a few specific points to the terminal screen. For example, the following example will list the data values for points between x = 1.66 GHz and x = 1.67 GHz for the PDC:

```
GBTIDL -> table, brange=1.66, erange=1.67
Scan:     79             W3OH 2005-06-28 +04 10 20.0
                         Ta
        GHz-LSR          YY
       1.6699992   -0.14701117
       1.6699962   -0.10967928
       1.6699931   -0.13311513
       1.669990    -0.10884448
           .              .
           .              .
           .              .
       1.6600114  -0.013062999
       1.6600084   0.023651161
       1.6600053   0.011518455
       1.6600022  -0.013523553
```

7.6 Annotating the Display

You can place text on the plot using the **annotate** procedure. This command takes three parameters: the x and y coordinates, and the text. You can also choose to include a color specifier and font size, as well as specify normalized coordinates (**/normal**). Example:

```
annotate, 6, 9, 'This is an annotation', color=!orange, charsize=2.0
```

7.7 Other Plotter Procedures

The following table lists some of the command line procedures relevant to the plotter. Full descriptions of these procedures are available in the GBTIDL User Reference or via the **usage** command.

Procedure	Action
show	Displays the spectrum
oshow	Display a spectrum as an overlay
gbtoplot	Used to plot arbitrary (x,y) values on the GBTIDL plotter.
chan, freq, velo, setxunit	Sets the X-axis units
setx, sety, setxy	Sets the X- and/or Y-axis scale
unzoom	Retrieve previous zoom settings
freex, freey	Auto scale one axis without affecting the range of the other.
freexy	Auto scale both axes
histogram	Toggle between histogram-style and connected-points
annotate	Place some text on the plot
crosshair	Toggle the crosshair cursor on/off
write_ascii	Write the data to an ASCII file
write_ps	Write the plot to a postscript file
zline	Toggle the zero-line on/off
bdrop, edrop	Hide channels at beginning/end of the spectrum
showregion	Turn on or off the display of baseline region boxes
click	Prompt user to click on the plot, and return info on the click location
clearannotations, clearvlines, clearoplots, clearoshows, clearovers, clearmarks, toggleovers	Clear various types of overlays
clear	Clear everything from the plotter
setabsrel, setframe, setveldef, setvoffset	Set the velocity definition and rest frame, and offsets
setmarker, vline	Place markers and lines on the plot
chantox, xtochan	Convert between X-axis units and channel number
freeze, unfreeze	Turn Auto Update off or on
reshow	Re-draw everything known to the plotter

7.8 Colors

GBTIDL has built-in color definitions in global variables called !black, !red, !orange, !green, !forest, !yellow, !cyan, !blue, !magenta, !purple, !gray, and !white. Many of the plotter commands take a color as an optional parameter. For example, the color of the spectral line can be changed like this:

```
show, color=!blue
```

8 Data Analysis

8.1 Using the Stack

The stack is a list of indices that can be used to gather scan numbers or record numbers to be used in a later operation, such as averaging. The stack system is modeled closely after the UNIPOPS commands.

To add entries to the stack, use the **addstack** command or the **appendstack** command. The **addstack** command adds a sequence of entries using parameters that describe the first entry, last entry, and increment. The **appendstack** command appends an array of indices to the stack. For example:

```
emptystack             ; clears the contents of the stack
addstack, 15           ; Add only index 15
addstack, 18, 21       ; Add indices 18 through 21
addstack, 22, 26, 2    ; Add indices 22 through 26 with an increment of 2. (22, 24, 26)
appendstack, [29, 35]  ; Adds indices 29 and 35 to the stack
```

The **tellstack** command lists the indices currently contained in the stack. The GBTIDL global variable `!g.acount` contains the total number of entries in the stack. The power of the stack will become more evident in the discussion on averaging data. For now, here is a simple example of using the stack to show spectrum headers for scans 6, 8, 10 and 12:

```
emptystack
addstack, 6, 12, 2
for i=0,!g.acount-1 do getnod, astack(i) & header & end
```

The following procedure gives an example of one way the stack could be put to use. The procedure averages Nod data identified by scan numbers listed in the stack. To use a procedure like this one, first populate the stack with the appropriate scan numbers then call the procedure.

```
pro myavg,_extra=extra
freeze
for i=0,!g.acount-1 do begin
    getnod,astack(i),plnum=0,units='Jy',_extra=extra
    accum
    getnod,astack(i),plnum=1,units='Jy',_extra=extra
    accum
endfor
ave
unfreeze
show
end
```

The following stack commands are available.

Command	Purpose
addstack	Adds a sequential list of indices to the stack, using addstack,begin,end,increment syntax
appendstack	Adds a single index or array of indices to the stack
astack()	Returns the value of a specific stack entry, given an index,
avgstack	Averages the records associated with the stack entries
delete	Removes a stack entry from the list
deselect	Removes indices from stack based on criteria such as source, polarization, and integration number.
emptystack	Clears the stack
liststack	Runs a **list** on records identified by the stack.
select	Adds indices to stack based on criteria such as source, polarization, and integration number.
tellstack	Shows the indices in the stack or returns all of the stack entries if no index is specified

8.2 Removing Baselines

GBTIDL uses "general orthogonal polynomials" in a least squares fit to determine baseline models. Currently GBTIDL does not support sinusoid or Fourier component models, but these may be added for a later release.

To remove a spectral baseline, you must first identify a line-free region of the spectrum to be fit. The region can be specified with either the **nregion** command, which allows you to specify the range by

typing the beginning and ending channels for each range, or with the **setregion** command, which allows you to select the baseline region on the plotter, using the mouse cursor.

You can specify the order of the polynomial with the **nfit** procedure, or provide it as a parameter in the baseline fitting routines. In either case, the value is stored and becomes the default for later baseline fits. You can view the baseline without subtracting it via the **bshape** procedure. When the baseline appears satisfactory, the **baseline** procedure can be used to subtract it. A typical baseline fitting session might then look like this:

```
nfit, 5         ; Specifies that a 5th order polynomial baseline will be fit
setregion       ; Specify baseline regions using the mouse
bshape          ; View the fitted baseline, but don't subtract it yet
nfit, 4         ; Specifies that a 4th order polynomial baseline will be used
bshape          ; View the new baseline fit, but don't subtract it
baseline        ; Subtract the most recent baseline fit
```

When a baseline is fit with either **bshape** or **baseline**, the baseline model itself can be stored in a global data container by setting the *modelbuffer* keyword. You can view the baseline model separate from the data as follows:

```
baseline, modelbuffer=5     ; Subtract the baseline and store the model in buffer # 5
show, 5                     ; Show the baseline model in buffer # 5
```

and the data could be restored to its original form by:

```
add, 5, 0                   ; Add baseline back to original spectrum to undo subtraction
```

After a baseline region is specified using the **setregion** procedure, a box is displayed indicating the region to be used in a baseline fit. The height of the box is twice the RMS of the data within the box, centered at the mean of the data within the box. These boxes can be removed using the **showregion, /off** command (the regions remain set, but are not displayed).

If you wish to subtract from spectrum B a baseline model derived from spectrum A, use this method:

```
getfs, 1                        ; Get spectrum A
baseline, modelbuffer=5         ; Fit and subract the baseline
getfs, 2                        ; Get spectrum B
subtract, 0, 5                  ; Subtract the old model
```

Here is a more sophisticated example of using various baseline features and commands.

```
getnod, 32                      ; Get some data
setregion                       ; Set a region to be fit
bshape, nfit=10                 ; Fit a 10th order polynomial
bmodel, nfit=2, modelbuffer=5   ; Use 2 coefficients to generate a new model
bmodel, nfit=5, modelbuffer=6   ; Use 5 coefficients
bmodel, nfit=10, modelbuffer=7  ; Use all 10
oshow,5, color=!yellow
oshow,6, color=!cyan
oshow,7, color=!green           ; Plot all three for comparison
subtract, 0, 6                  ; Subtract the 5th order fit
```

8.3 Averaging Data

GBTIDL uses an accumulator to average data. For example:

```
sclear           ; Clears the default global accumulator
get, index=1     ; Get record # 1
accum            ; Put the data in the accumulator
get, index=2     ; Get record # 2
accum            ; Adds the data to the accumulator
ave              ; Averages data in the accumulator and places result in PDC
```

The **sclear** command clears the *accum* buffer to ensure it starts empty. The result of the average is then stored in the PDC unless otherwise stated.

The above example uses the default accumulator buffer. There are 4 accumulator buffers numbered 0, 1, 2, and 3 so you can perform up to 4 different averages simultaneously. These are useful, for example, when accumulating data from two polarizations simultaneously, as shown in the following script:

```
sclear, 1          ; Clear the 1st accum buffer
sclear, 2          ; Clear the 2nd accum buffer
for i=10,15 do begin
 getfs, i, plnum=0
 accum, 1           ; Put data in 1st buffer
 getfs, i, plnum=1
 accum, 2           ; Put data in 2nd buffer
end
ave, 1             ; Average data in 1st buffer
copy, 0, 10
ave, 2             ; Average data in 2nd buffer
copy, 0, 11
show, 10
oshow, 11
```

Note that the IDL code in the above example works only if it is stored as a script, not interactively, because the *for* loop is split over several lines without the IDL line continuation characters & and $.

When the **ave** command is issued, the contents of the *accum* buffer are cleared unless the **noclear** keyword is set. So, if you wish to view intermediate results in an ongoing average, you must specify that the buffer should not be cleared:

```
sclear
get, index=1
accum
get, index=2
accum
ave, /noclear         ; The accum buffer is NOT cleared here
get, index=3
accum
ave                   ; The accum buffer IS cleared here
```

It is also possible to use the stack when averaging data by using the **avgstack** command. In the following example, the stack is used to identify records in the data file, and these are averaged.

```
addstack, 25         ; Add index 25 to stack
addstack, 30, 39     ; Add indices 30 through 39 to stack
avgstack             ; Average the stack (data in records 25, 30-39)
```

In the following example, we select some data associated with the "LL" polarization and average them.

```
emptystack                                                  ; Start with an empty list
select,source='W3OH',scan=[177,178],pol='LL',cal='F'
tellstack
liststack
delete,4                                                    ; Remove record 4 from the list
avgstack                                                    ; Average the three scans in the stack
show
```

8.4 Averaging Data not Aligned in Frequency

Suppose you wish to average spectra that overlap in frequency but are not exactly aligned. You must use **fshift** to determine the shift needed to align the spectra, apply that shift using **gshift**, and then add the spectra to the accumulator and average. For example:

```
getps, 30
accum              ; Accumulate first spectrum, no alignment needed yet
getps, 32
fs = fshift()      ; Determine the shift to align scan 32 with the spectrum in
                   ;  the accumulator
gshift,fs          ; Apply the shift to scan 32 in the PDC
accum              ; Add the result to the accumulator
getps, 34
gshift, fshift()   ; All in one line, shift 34 to align with the accumulator
accum
ave
```

It is also possible to align spectra on the basis of velocity using **vshift**, or using the current x-axis units using **xshift**.

8.5 Smoothing Data

GBTIDL provides users with 3 different smoothing options: boxcar, Gaussian, and hanning. In each case it is possible to use "decimation", which means that every *n-th* channel will appear in the smoothed spectrum, n being determined by the smoothing parameters. Boxcar smoothing requires a parameter to specify the width of the boxcar. The **gsmooth** feature convolves the data with a Gaussian of width $\sqrt{newres^2 - origres^2}$, where *newres* is the new resolution given by the user in units of channels. The hanning smooth uses a 3-channel hanning filter. Examples:

```
getps, 25               ; Get some data into the PDC
boxcar, 4               ; 4-channel boxcar smooth, no decimation
getps, 25               ; Get some data into the PDC
boxcar, 2, /decimate    ; 2-channel boxcar with decimation (keeps every other channel)

getps, 25               ; Get some data into the PDC
gsmooth, 4, /decimate   ; Smooth to 4 channels & decimates (keeps every 4th channel)
\end{small}
getrec, 1               ; Get some data
hanning                 ; Apply hanning smooth and show the result
```

8.6 Fitting Gaussian Profiles

The procedure **fitgauss** is used to fit Gaussian profiles to spectral line data. Since a Gaussian function approaches zero away from the line center, you get the best results by subtracting a baseline from the data prior to using **fitgauss**. In general the procedure for Gaussian fitting is as follows:

- Subtract a baseline from the spectrum of interest.
- Using the plotter, zoom in to a region near the lines to be fit.
- Run the **fitgauss** procedure
  - Mark the line to be fit using the left mouse button. Only the channels selected will be included in the fitting algorithm. By selecting carefully, it is possible to have the procedure ignore any nearby lines or even fit one among blended lines.
  - Using the **middle** mouse button, click first on the peak of the line to be fit, and then middle-click again on the half-power point. These two clicks specify the initial guesses for line height, width, and center used by the Gaussian fitter. To fit multiple profiles simultaneously, continue to click the middle mouse button to mark additional lines.
  - When all lines have been marked, click the **right** mouse button to do the fit.

To retain the continuum level in a fit of absorption lines, the following recipe can be applied:

- Determine the continuum level of the source.
- Fit and subtract a baseline.
- Fit the absorption line with a Gaussian and save the model using the modelbuffer parameter.
- Add the continuum as a bias to both the data and the model.

For example, suppose we wish to fit an absorption line on a 1.5 Jy continuum source, and display the fit as an overlay.

```
setregion                    ; Set the baseline region
nfit, 3                      ; Plan to fit a 3rd order polynomial baseline
baseline, modelbuffer=3      ; Fit and subtract the baseline.  Continuum is also subtracted.
fitgauss, modelbuffer=10     ; Fit the Gaussian and store the model in buffer 10
bias, 1.5                    ; Add the continuum level back to the data
copy, 0, 5                   ; Store the data in buffer 5
copy, 10, 0                  ; Copy the model to buffer 0
bias, 1.5                    ; Add the continuum level to the model
copy, 0, 10                  ; Return the model to buffer 10
copy, 5, 0                   ; Return the data to buffer 0
oshow, 10, color=!orange     ; Overlay the model on the data
```

8.7 Introduction to Flagging and Blanking Data

RFI and other faults that cause intermittent or frequency-dependent bad data make it necessary to be selective when operating on a data set. Bad data can be addressed with a combination of flagging and blanking. Flagging is the process of assigning a set of rules for marking bad data. Blanking is the process of applying these rules to the data, and replacing the flagged data with a special blanking value. See the "More About Flagging Data" section in the Appendix for more information.

The most common purpose of flagging and blanking is to identify data to be excluded from a calibration or averaging operation. As such, flagging is usually applied to raw data and data that have not yet been averaged.

In GBTIDL, the special value for blanked data is the IEEE not-a-number (NaN). Many native IDL procedures already recognize that value and treat it appropriately. So, operations such as fitting and averaging will ignore NaN values. As an example of the special handling of blanked values, consider the **show** command. It handles the special values by putting gaps in the plotted spectrum at the locations of blanked data. The **stats** procedure simply ignores any blanked channels in computing the statistics. The **hanning** procedure blanks channels in the smoothed spectrum whose constituent channels are themselves blanked. In general, procedures know how to take the appropriate action when they encounter blanked data, and this action varies depending on the procedure.

Blanking is automatically applied to data when it is read into memory using the calibration or I/O procedures such as **get, getfs, getps**, etc. Blanking can also be applied by using the **replace** command.

As an example, suppose you have a spectrum displayed in the plotter and you would like to blank bad data in channels 500 to 525. The following command will perform the task.

```
replace, 500, 525, /blank    ; Blanks the range of channels from 500 to 525
```

Flagging is different from blanking in that flagging does not change the data in a data container. Instead, flagging commands are associated with data on disk, and describe which of those data should be blanked when it is read with the GBTIDL I/O routines. The flagging commands are stored in a separate file from the data file, so you can unflag data or selectively ignore or apply certain flagging rules without changing the data on the disk or in memory.

Examples of setting flag rules, changing flag rules, and blanking data:

You know your data are bad in channels 500 to 525 and 1000 to 1100 for scan 11 but just in plnum=1 and ifnum=2. However, the data in the two channel ranges are bad for different reasons. The flags would be set and the data blanked like this:

```
flag, 11, plnum=1, ifnum=2, $                  ; Flag and label "rfi"
      chans="500:525", idstring="rfi"
flag, 11, plnum=1, ifnum=2, $                  ; Flag and label "acs_glitch"
      chans="1000:1100", idstring="acs_glitch"
getfs, 11, plnum=1, ifnum=2                    ; Flagged data are now blanked in the PDC
```

Notice the use of *idstring* to document the reason a particular flag is being used.

If you have set up flagging rules but wish to ignore them when reading the data, the following command will retrieve the data without blanking:

```
getfs, 11, plnum=1, ifnum=2, /skipflag      ; No flags are applied
```

If you want to flag the first integration of a range of scans because you suspect the telescope was still settling and not on target:

```
flag, scanrange=[6,10], intnum=0, idstring="first int"  ; Flags integration 0 of scans 6-10
```

To view all the flag rules that have been set, use the **listflags** command:

```
listflags     ; Produces a list of all the flag rules established
```

To remove a flag rule, use the **unflag** command. This works by either providing the *idstring* attached to a flag or an integer matching an ID number as shown by **listflags**:

```
unflag, "first int"  ; Unflags the rule with the id string "first int"
```

More examples can be found in the "More About Flagging Data" section in the Appendix.

8.8 Statistics

Statistics are available from the **stats** procedure. If **stats** is given no parameters, you must specify the range over which statistics are calculated using the mouse cursor. Otherwise, **stats** can take two parameters, indicating the begin and end values for the range, in units currently displayed on the plotter. The */chan* keyword can be used if you want to give the range in channels regardless of the current plotter units. For example:

```
getrec, 1                  ; Get some data
stats                      ; Prompt user for the range using the mouse
stats, 1420.0,1420.1       ; Show stats over the specified range
stats, /full               ; Show stats over the full spectrum
stats, /full, ret=mystats  ; Return statistics to the IDL data structure called mystats
print, mystats.mean        ; Prints the mean value stored in the mystats data structure
print, mystats.rms         ; Prints the rms value stored in the mystats data structure
stats, 0, 99, /chan        ; Show stats for the first 100 channels
```

Here is an example of the output of the **stats** command:

```
GBTIDL -> stats
Click twice to define stats region
    Chans    bchan    echan        Xmin        Xmax        Ymin        Ymax
    13661    10692    24352      1.6103      1.6155     -4.0177      14.259

                      Mean      Median         RMS    Variance        Area
                  -0.14350   -0.051825     0.55484     0.30785 -0.00074783
```

8.9 Using the Select and Find Features

8.9.1 Select

The **select** procedure in GBTIDL is used to search and select records from the input data set and add indices of the matching entries to the stack. To locate the relevant records, **select** uses the contents of the GBTIDL index file. The parameters for the **search** procedure are the same as those for the **get** procedure given in section 6.2. The procedure **listcols** can be used to list all parameters available for searching. Note that in this procedure, like all IDL procedures, the parameter names do not need to be typed in their entirety, only enough characters to uniquely identify the parameter are necessary.

To select all records associated with a given source name:

```
select, source='3C286'
```

Multiple parameters are combined with a logical AND, so the following command selects all 3C286 records between scans 100 and 119:

```
select, source='3C286', scan=seq(100:119)
```

To select specific integer values, use an array as follows:

```
select, source='3C286', scan=[100,102,104,106]
```

The syntax for selections depends on the data type that is being selected, as shown in the following examples.

Integer Searches

```
select, index=10                        ; Selects one index
select, index=[10,14,17,18]             ; Selects a list of indices
select, index='10:15,20:23'             ; Selects the given ranges
select, index=':30'                     ; Selects indices less than 30
```

Float Searches

```
select, tsys='33.26'                    ; Selects values between 33.255 and 33.265
select, tsys='33.0:38.0'                ; Selects the range 33-38 K
select, tsys=':45.0'                    ; Selects based on Tsys < 45.0K
select, tsys=33.26                      ; Selects values that are exactly 33.26, rarely useful
```

String Searches

```
select, source='NGC1068'                ; Select based on single string value
select, source=['NGC1068', 'NGC1069']   ; Select from a list of strings
select, source='NGC*'                   ; Wildcards allowed at beginning and end of string
```

8.9.2 Find

The **find** procedure and the related procedures **setfind**, **clearfind**, and **listfind** (each described below) use **select** in a way that has been designed to mimic some of the features of the CLASS **find** command. The **find** command is particularly useful if you want to repeat the same or slightly modified selection. Each use of **find** first clears the stack (unlike **select**) unless the **/append** keyword is used.

- **setfind:** Used to set specific selection criteria. Once set, they remain set until cleared using **clearfind**.
- **find:** Used to place the entries specified by the **setfind** command into the stack.
- **clearfind:** Used to clear the current setfind selection criteria.
- **listfind:** Used to list a specified selection parameter or all selection parameter values used by **find**. This allows you to tell the value of one or all of the selection parameters used by **find**.

Examples:

First define the initial selection criteria:

```
GBTIDL -> setfind, 'scan', 80, 82           ; Select scans 80 through 82
GBTIDL -> find                              ; Add the selection to the stack (See 8.1)
Indices added to stack : 288
GBTIDL -> listfind                          ; Show current selection parameters
All set FIND parameters for LINE mode
SCAN 80:82
```

Then refine them:

```
GBTIDL -> setfind, 'polarization', 'XX'     ; Select only the XX polarization
GBTIDL -> find                              ; Update the stack so it only contains scans 80-82 with
     Indices added to stack : 144
GBTIDL -> listfind                          ; Show current selection parameters
     All set FIND parameters for LINE mode
     SCAN 80:82
     POLARIZATION XX
```

Refine them again:

```
GBTIDL -> setfind, 'int', 3                  ; Select only integration 3
GBTIDL -> find                               ; Update stack to only contain indices that satisfy all
     Indices added to stack : 24
GBTIDL -> listfind                           ; Show current selection parameters
     All set FIND parameters for LINE mode
     SCAN 80:82
     POLARIZATION XX
     INT 3
```

Change your mind and decide to include integration 4 also:

```
GBTIDL -> setfind, 'int', 4, /append         ; Use the /append keyword to add data
GBTIDL -> find                               ; Add the 4th integration indices to stack
     Indices added to stack : 48
GBTIDL -> listfind                           ; Show new selection parameters
     All set FIND parameters for LINE mode
     SCAN 80:82
     POLARIZATION XX
     INT 3,4
```

8.10 Mapping

GBTIDL currently does not support mapping. There is a mechanism for exporting SDFITS data into classic AIPS. Contact your GBT support person for details.

8.11 Other Analysis Procedures

The following table lists additional analysis commands that may be useful. More information on these, and other commands, can be found in the User Reference or by using the **usage** command.

Procedure	Action
clip, datamin, datamax	Truncate spectrum to a min and max data value
decimate	Decimate the spectrum by paring channels
gconvol	Convolve the spectrum in the PDC with an array
gfft	FFT or inverse FFT the spectrum
ginterp	Interpolate across blanked channels
gmeasure	HI profile fitting procedure
gmoment	Caclulate first 3 moments
invert	Flip the data end-to-end
molecule	Show molecular transition frequencies on the plotter
powspec	Compute power spectrum
recomball	Plot the H alpha, beta, gamma; He alpha, beta, and C alpha recombination lines
recombc	Compute and plot frequencies of Carbon recombination lines
recombhe	Compute and plot frequencies of Helium recombination lines
recombh	Compute and plot frequencies of Hydrogen recombination lines
recombn	Compute and plot frequencies of Nitrogen recombination lines
recombo	Compute and plot frequencies of Oxygen recombination lines
replace	Replace bad data values
resample	Resample the spectrum in the PDC at the new interval

9 Saving and Retrieving Data

To save data to disk, first specify the name of the output data file using the **fileout** command. The default file is called "GBTIDL_keep.fits". The name of the data file must end in ".fits". Spectra can be

written to this file using either the **keep** or **nsave** command.

9.1 keep

The **keep** command saves a spectrum to the output file, appending the data to the end of that file. For example:

```
getnod,30                ; Get some data
fileout,'mysave.fits'    ; Open an output file
keep                     ; Writes the PDC to the file
getnod,32                ; Get some more data
keep                     ; Write the PDC to file
getfs,48,/nofold         ; Get some more data
keep                     ; Writes the PDC to file
keep,5                   ; Writes info in DC 5 to file
```

9.2 nsave

The **nsave** feature allows you to store data in a file and attach an identifing number (the *nsave* value) to that entry so that the data can be overwritten or retrieved according to the *nsave* value. The utility essentially gives you access to an unlimited number of storage slots on disk, somewhat like the 16 global buffers kept in memory.

The following GBTIDL procedures are relevant to the *nsave* features.

Command	Purpose
fileout	Set the output file; note this can be used for both **keep** and **nsave**
nget	Retrieve a spectrum with a given *nsave* value
nsave	Store a spectrum to disk with the given *nsave* value
sprotect_off	Enable overwrite permission
sprotect_on	Disable overwrite permission

The following sequence shows how to store and retrieve a spectrum using the **nsave** feature:

```
fileout, 'mynsave.fits'     ; Open a file for writing
getrec, 10                  ; Get some data
nsave, 101                  ; Store it to the keep file
scale, 5                    ; Perform some operations
nsave, 102                  ; Store it with a different nsave value
nget, 101                   ; Retrieve the spectrum at nsave=101
```

The next example shows a more sophisticated *nsave* example. Here each *nsave* entry stores a calibrated integration from a scan. The example demonstrates how the *nsave* values can be overwritten and each *nsave* entry has an attached meaning to the data, for example nsave=1002 is the data associated with scan=100, int=2. As the calibration of this integration is refined, the spectra are simply stored back into that *nsave* slot.

```
fileout, 'mynsave.fits'         ; Open a file for writing
for i=0,5 do begin $            ; Store each integration
     getnod, 100, int=i & $
     nsave, 1000+i & $
endfor

nget, 1002                      ; Retrieve one of the entries
```

```
bias, 0.1                       ; Do some work on it
nsave, 1002                     ; And reinsert it into the file

for i=0,5 do begin $            ; Now execute a loop to average all
     nget, 1000+i & $           ; the data including the processed
     accum & $                  ; integration.  This loop could be
     endfor                     ; made into a separate procedure.
ave
```

9.3 Retrieving Data from the Output File

To retrieve data saved using GBTIDL, it is possible to open the file as an input file and use the **get** and **getrec** commands. The following example illustrates.

```
getnod, 101             ; Get a spectrum from a previously defined input file
fileout, 'mydata.fits'  ; Set the output file name
keep                    ; Store the spectrum in record 0 of the keep file
getnod, 103             ; Get more data
keep                    ; Store the next spectrum in record 1
fileout, 'KEEP.fits'    ; Close mydata.fits and open a new output file
filein, 'mydata.fits'   ; Reopen mydata.fits it as an input file
getrec, 0               ; Retrieve the first entry
```

Alternatively, data can be retrieved directly from the output file using **kgetrec** or **kget**.

```
getnod, 101             ; Get a spectrum from a previously defined input file
fileout, 'mydata.fits'  ; Set the output file name
keep                    ; Store the spectrum in record 0 of the keep file
getnod, 103             ; Get more data
keep                    ; Store the next spectrum in record 1
kget, scan=101          ; Retrieves scan 101 from the output data file and places it in the PDC
```

The **kget** command uses the same selection parameters as the **get** procedure.

10 Writing Your Own Procedures

GBTIDL is designed to allow you to write your own procedures easily. The best approach to writing your own procedures is to start by looking at the code of a similar existing procedure. All the code in GBTIDL is available for your perusal in the Green Bank and Charlottesville installations at `/home/gbtidl/release/gbtidl`. All of the NRAO-developed or modified code can be found in the *pro* subdirectory, user-contributed code can be found in *contrib*, and IDL code from other sources can be found in the *lib* subdirectory.

To write custom procedures, you should become familiar with IDL programming, and with the data container structure. Here is a simple example of a procedure to use as a template. This example scales the data in the spectrum by a factor given by the user.

```
pro myscale,factor
     tmp_data = getdata()
     tmp_data = tmp_data * factor
     setdata, tmp_data
     if !g.frozen eq 0 then show
end
```

Suppose the code is stored in a file is called *myscale.pro*. To access the function, do this:

```
.compile myscale.pro  ; Compile the program
show                   ; Show the data
myscale, 2             ; Scale the data by a factor of 2
```

That's it!

You can put procedures in the directory from which you are running GBTIDL, or in a special subdirectory off of your home directory called *gbtidlpro*. In case there are procedures with identical names in your IDL path, the directories will be searched in the following order: first the current directory, then *$HOME/gbtidlpro*, then the GBTIDL installation directories, and finally the IDL installation itself. If the file isn't in one of these directories, you will need to specify the path when compiling it:

```
.compile /users/aeinstein/mypros/myscale.pro
```

A The !g Structure

The following table describes the contents of the global structure `!g` used by many GBTIDL routines. Using this structire improves the interface of many GBTIDL routines by not requiring that all parameters be specified each time a procedure is run. To view the structure of `!g`, type in GBTIDL:

```
GBTIDL -> help, !g, /struct
```

Name	Type	Default	Description
version	string	'2.8'	A version identifier for this version of gbtidl
s	spectrum data container array	empty spectrum	16 spectrum data structures
c	continuum data container array	empty continuum	16 continuum data structures
lineio	io_sdfits_line object	not connected	An io_sdfits_line object
contio	io_sdfits_cntm object	not connected	An io_sdfits_cntm object
lineoutio	io_sdfits_writer object	GBTIDL_KEEP.fits	An io_sdfits_writer object
line_filein_name	string		The last argument to filein, Sdirin, or online in "line" mode
cont_filein_name	string		The last argument to filin or dirin in "cont" mode
line_fileout_name	string	GBTIDL_KEEP.fits	The last argument to fileout
frozen	long integer	0	When true, the plotter is not updated (frozen) after !g.s[0] or !g.c[0] is changed.
sprotect	long integer	1	When true, you can not overwrite an NSAVE in fileout. "s" refers to "save".
line	long integer	1	When true, !g.s and !g.lineio are used, else =!g.c and !g.contio.
plotter_axis_type	long integer	1	0=Channels, 1=Frequency, 2=Velocity
has_display	long integer	1	Used to distinguish sessions that cannot use a plotter
interactive	long integer	1	Will be 0 for non-interactive sessions (e.g. cron jobs)
background	long integer	!black	Default plotter background color
foreground	long integer	!white	Default plotter foreground color
showcolor	long integer	!red	Default color for "show"
oshowcolor	long integer	!white	Default color for "oshow"
crosshaircolor	long integer	!gree	Default crosshair color
zlinecolor	long integer	!green	Default zero-line color
markercolor	long integer	!green	Default marker color
annotatecolor	long integer	!green	Default color for annotate
oplotcolor	long integer	!white	Default color for oplot
vlinecolor	long integer	!green	Default vline color
zoomcolor	long integer	!cyan	Default color for zoom box
gshowcolor	long integer	!white	Default color for gshow
gausstextcolor	long integer	!white	Default color for gshow text
highlightcolor	long integer	!cyan	Default color for highlighted data (fitgauss, gmeasure)
colorpostscript	long integer	1	If not true, generated postscript will be black and white, ignoring any colors.
astack	pointer	0	The stack contents, as an array of long integers. Initially 5120 elements but this is extended as needed.
acount	long integer	0	Number of values of astack actually used.
accumbuf	Array of 4 accum_struct structures	empty accum_struct	An array of 4 accum buffers used by accum, ave, et al.
regions	long integer array	All -1	2D with shape [2,100] array holding regions to use in fitting. Actual regions in use are given by nregion.
nregion	long integer	0	Total number of regions in regions that are in use currently. [*,0:(nregion-1)]
regionboxes	long integer	0	When true, the region boxes on the plotter are persistent as new plots are shown
nfit	long integer	-1	Most recent polynomial order for a baseline fit (bshape and baseline).
polyfit	double array	0	2D array holding the parameters of the most recent baseline fit. The shape is [4,51] and [*,0:nfit] are in use at any time.

polfitrms	double array	0	1D array holding RMS for each of the (nfit+1) polynomials in the most recent baseline fit. Shape is [51] and [0:nfit] are in use at any time.
Gauss	Gauss structure	Gauss defaults	The structure used by the Gaussian fitting routines
printer	string	PRINTER environment variable from the unix shell	Identifies the printer to use.
tau0	float	0.0	Zenith tau
ap_eff	float	0.7	Aperture efficiency
nsave	long integer	-1	Most recent NSAVE used.
user_list_cols	string		Comma-separated list of index column names. Used by list and liststack when /user is set.
find	find_struct structure	''	The find structure. Fore use with find, setfind, and clearfind.
molecules	Array of 4000 molecule_struct structures	empty	A structure used by molecule.
nmol	long integer	0	Number of elements of molecules in use.

B Tips on Using Data Containers for Experts

There are 16 global data containers, or buffers, numbered 0 through 15. Data container 0 is also called the primary data container, or PDC for short. If you find you need more than 16 buffers, one option is to use the **nsave** facility, which allows you to store an arbitrary number of data containers in a disk file. Alternatively, you can store data containers as IDL variables. If you choose to store data contrainers in IDL variables, there are a few procedures you should be aware of:

- **data_new**: Create a new data container.
- **data_copy**: Copy a data container.
- **data_free**: Free the pointers in a data container or array of data containers.
- **set_data_container**: Copy a data container stored as a variable into one of the 16 global buffers.

Check the reference pages or look at code examples for help on using these procedures. Make sure that when you create a new data container (either by **data_new** or **data_copy**) you free the pointers using **data_free** when you are done, otherwise memory will be leaked.

Be sure to avoid this mistake when using data containers:

```
GBTIDL -> mydc = !g.s[0]
     ; ... you do stuff to mydc here
     ; ... you think you are done, so you free it
GBTIDL -> data_free, mydc
```

The mistake here is that the initial assignment copies the value of the pointer, not the array that the pointer refers to. So, any changes to the data array through mydc will also change the data array in **!g.s[0]** because they use the same pointer. More importantly, the **data_free** at the end will also free the pointer in **!g.s[0]**, likely bringing GBTIDL to its knees.

Instead, use **data_copy**:

```
GBTIDL -> data_copy, !g.s[0], mydc
GBTIDL -> set_data_container, mydc              ; Resets index 0 with the contents of mydc
GBTIDL -> data_copy, !g.s[1], mydc
GBTIDL -> set_data_container, mydc, index=1     ; Resets index 1 with the contents of mydc
GBTIDL -> data_free, mydc
```

This example illustrates the use of **set_data_container** to copy a user-named data container into the global data container. It is not necessary to use **data_free** before calling **data_copy** because **data_copy** takes care of all pointer maintenance in the output data container without leaking memory.

Also, be aware that when global values are used as parameters to functions or procedures, IDL passes those values by value and not by reference. So if you send a DC from `!g` to a procedure or function, all changes you make to that DC will remain local to that function, and will not be retained in the global variable.

If you need to work with an array of data containers here is one way you might do that:

Suppose you want to run **getfs** on scans 50 through 100 and defer saving the data to the output file until the end. The step where the data are written to disk will be much faster if it can all be done at once, but it does mean that all 51 spectra will be in memory by the end of this operation so you should consider whether they will all fit in memory at the same time.

```
dcarr = replicate({spectrum_struct},51)   ; Create un-initialzed array of 51 data containers
freeze                                    ; Turn off the plotter's auto-update feature
for i=50, 100 do begin                    ; Loop over the scan numbers
     getfs, i
     tmp = dcarr[i-50]                    ; Copy that to dcarr (this is the tricky step)
     data_copy, !g.s[0], tmp
     dcarr[i-50] = tmp
endfor
putchunk, dcarr                           ; Save it
data_free, dcarr                          ; Free up the memory
```

The *tmp* variable is used in the *for* loop because of the aforementioned issue of IDL passing elements to functions and procedures by value and not by reference. So we assign to *tmp* the specific element of *dcarr* that we want to modify. That gets a copy of everything, including the pointer. Inside **data_copy**, *tmp* is modified and since *tmp* in this case is passed by reference (because it is not a global value and it isn't an array element), changes to *tmp* will be seen outside of **data_copy**. Once **data_copy** returns, the values in *tmp* (including the now valid pointer containing the copy of the data array) will be copied to *dcarr*. It is not necessary or desirable to use **data_free** on *tmp* because that would also free the copy of that pointer in *dcarr*. That pointer is freed at the end. Be sure and free up all of the pointers that you create this way so that memory is not leaked.

C Contents of the Spectrum Data Container

The following table describes the contents of the spectrum data container. It is closely modeled on the contents of SDFITS files.

Name	Type	Default	Index	Description
data_ptr	pointer to float array	0		The data array. Must be dereferenced to get data values.
zero_channel	double	NaN		The zerochan value from the SDFITS file. Will be a finite value only for ACS data.
units	string	counts		The units of the data.
source	string		source	Source name
observer	string			Observer's name
projid	string		project	Project ID
scan_number	long integer	0	scan	The scan number
procseqn	long integer	0	procseqn	The procedure sequence number
nsave	long integer	-1	nsave	The nsave location this has been saved to. Will only be = 0 if NSAVE was used to save this to the output file.
procedure	string		procedure	The name of the observing procedure
procsize	long integer	1		Total number of subscans expected for this scan (procedure)
switch_state	string			The type of switching used during this procedure.
switch_sig	string			The switching signal scheme used during this procedure.
sig_state	long integer	1	sig	If true (1) this is the signal state, otherwise the reference state
cal_state	long integer	0	cal	If true (1) the cal was on, otherwise it was off
caltype	string			The type of cal. Currently either HIGH or LOW.
integration	long integer	0	int	The integration number (starting from 0)
if_number	long integer	0	ifnum	The IF number. For GBT data, the IF_NUMBER will correspond to the same IF in other data containers if the setup was identical.
obsid	string		obsid	GBT manager scanId, from the GBT GO FITS file.
backend	string			Backend name
frontend	string			Frontend (receiver) name
exposure	double	0.0	exposure	Effective integration time in seconds
duration	double	0.0		The clock time spent collecting this data, including blanking time, in seconds
tambient	float	0.0		The ambient temperature in K
pressure	float	0.0		The pressure in Pa
humidity	float	0.0		The humidity fraction 0.1
tsys	float	1.0	tsys	System temperature in K
mean_tcal	double	1.0		The mean Tcal value across the entire bandpass in K
tsysref	float	1.0		System temperature of reference data used by this data in K
telescope	string	NRAO_GBT		Name of the telescope
site_location	3 doubles	Location of the GBT		(East longitude in degrees, latitude in degrees, elevation in meters)
coordinate_mode	string	RADEC		The type of coordinate for the pointing direction (inferred from CTYPE2 and CTYPE3). Possible choices are RADEC, GALACTIC, HADEC, AZEL, and OTHER.
polarization	string		polarization	The polarization
polarization_num	long integer	0	plnum	The polarization number. Counts from 0 for each scan.
feed	long integer	0	feed	The feed name as known at the telescope.
srfeed	long integer	0		The switching feed name as known at the telescope.
feed_num	long integer	0	fdnum	The number of this feed. Counts from 0 for each scan.
feedxoff	double	0.0		Beam offset for the cross-elevation axis, in degrees
feedeoff	double	0.0		Beam offset for the elevation axis, in degrees
sampler_name	string		sampler	The name of the sampler (a GBT-specific term)
bandwidth	double	0.0	bandwidth	Total bandwidth in Hz
observed_frequency	double	0.0		The observed (sky) frequency in Hz at the reference_channel
sideband	string			The sideband (U or L)
equinox	double	2000.0		The equinox, in years, of the longitude and latitude axis values, when appropriate.
radesys	string	FK5		The equitorial coordinate system when appropriate, e.g. FK5, FK4, GAPPT.
date	string	current date	[dateobs]	Date (YYYY-MM-DD), along with utc, corresponding to mjd

utc	double	current time	[dateobs]	UTC seconds since start of date. Corresponds to mjd
mjd	double	from current date and time		Modified Julian Date at mid-point of integration in days
timestamp	string	default	timestamp	The timestamp given to the scan when it was taken. YYYY_MM_DD_HH:MM:SS. This can be used in data selection when there are repeated scan numbers.
frequency_type	string	TOPO		Description of the frequency axis. From CTYPE1 Recognized values are TOPO, LSR, LSD, GEO, HEL, BAR, and GAL.
reference_frequency	double	reference channel		Frequency, in Hz, at the reference_channel
reference_channel	double	n_elements(data_ptr)/2 + 1		The reference channel
frequency_interval	double	1.0	freqint	Spacing in Hz between adjacent channels: f(i+1)-f(i)
frequency_resolution	double	1.0	freqres	The spectral resolution of one channel, in Hz.
center_frequency	double	0.0	centfreq	The frequency in Hz at the center channel, which may not be reference_frequency. This is used in the index file and is available for selection.
longitude_axis	double	0.0	longitude	The longitude pointing direction in degrees in coordinate_mode at equinox
latitude_axis	double	0.0	latitude	The latitude pointing direction in degrees in coordinate_mode at equinox
target_longitude	double	0.0	trgtlong	The target (source) longitude pointing direction in degrees in the same coordinate system as longitude_axis. From the GO FITS file.
target_latitude	double	0.0	trgtlat	The target (source) latitude pointing direction in degrees in the same coordinate system as latitude_axis. From the GO FITS file.
velocity_definition	string	RADI-OBS		SDFITS VELDEF keyword
frame_velocity	double	0.0		True (relativistic) velocity of the doppler tracked frame with respect to the telescope in m/s
lst	double	from current time	lst	The LST in seconds corresponding to UTC on date given the site_location
azimuth	double	0.0	azimuth	Azimuth in degrees
elevation	double	0.0	elevation	Eevation in degrees
subref_state	integer	1	subref	Subreflector state when subreflector nodding; 0=moving, 1=first position, -1=second position
line_rest_frequency	double	0.0	restfreq	Rest frequency of line of interest in Hz
source_velocity	double	0.0	velocity	Velocity of the source in m/s in the reference frame and definition given by velocity_definition
freq_switch_offset	double	0.0		The offset of the reference switching state to the signal state in Hz for calibrated data

The column in this table labeled *Index* shows the name that each field is known by in the index file. These are the names returned by the **listcols** command. If the *Index* entry is blank, that field is not represented in the index file and is not available for selection. Note that the `dateobs` index name corresponds to the full DATE-OBS value from the SDFITS file, which combines the information in the date and utc fields from the data container.

The `zero_channel` value is only relevant for data from the spectrometer (ACS). When the N lags are transformed from lags to frequency space, there are N+1 unique frequencies. The data values are taken from channels 1 through N and the 0 value is stored in the ZEROCHAN column of the SDFITS file. The value is not used by any core GBTIDL procedures but is included here for experienced users who wish to make use of it.

D Contents of the Continuum Data Container

The following table describes the contents of the continuum data container. It is closely modeled on SDFITS files.

Unlike the spectral line DC, the continuum DC contains several pointers to arrays (date, utc, mjd, longitude_axis, latitude_axis, lst, azimuth, elevation and subref_state) in addition to the data array. The length of each of these arrays is the same as the data array. Each element in these arrays corresponds to one integration for that sampler and switching state. The frequency at the center of the bandpass is given by the observed_frequency field and is always a topocentric frequency.

Name	Type	Default	Index	Description
data_ptr	pointer to float array	0		The data array. Must be de-referenced to get data values.
units	string	counts		The units of the data.
source	string		source	Source name
observer	string			Observer's name
projid	string		project	Project id
scan_number	long integer	0	scan	The scan number
procseqn	long integer	0	procseqn	The procedure sequence number
procedure	string		procedure	The name of the observing procedure
procsize	long integer	1		Total number of subscans expected for this scan (procedure)
switch_state	string			The type of switching used during this procedure.
switch_sig	string			The switching signal scheme used during this procedure.
sig_state	long integer	1	sig	If true (1) this is the signal state, otherwise the reference state
cal_state	long integer	0	cal	If true (1) the cal was on, otherwise it was off
caltype	string			The type of cal. Currently either HIGH or LOW.
integration	long integer	0		The integration number (starting from 0)
if_number	long integer	0	ifnum	The IF number. Counts from 0 for each scan. For GBT data, the same IF_NUMBER will correspond to the same IF in other data containers so long as the setup was identical.
obsid	string		obsid	GBT manager scanId, from the GBT GO FITS file, currently unused
backend	string			Backend name
frontend	string			Frontend (receiver) name
exposure	double	0.0		Effective integration time in seconds
duration	double	0.0		The clock time spent collecting this data, including blanking time, in seconds
tambient	float	0.0		The ambient temperature in K
pressure	float	0.0		The pressure in Pa
humidity	float	0.0		The humidity fraction 0.1
tsys	float	1.0		System temperature in K
mean_tcal	double	1.0		The mean Tcal value across the entire bandpass in K
tsysref	float	1.0		System temperature of reference data used b56y this data in K
telescope	string	NRAO_GBT		Name of the telescope
site_location	3 doubles	Location of the GBT		(East longitude in degrees, latitude in degrees, elevation in meters)
coordinate_mode	string	RADEC		The type of coordinate for the pointing direction (inferred from CTYPE2 and CTYPE3). Possible choices are RADEC, GALACTIC, HADEC, AZEL, and OTHER.
polarization	string		polarization	The polarization
polarization_num	long integer	0		The polarization number. Counts from 0 for each scan.
feed	long integer	0		The feed name as known at the telescope.
srfeed	long integer	0		The switching feed name as known at the telescope.
feed_num	long integer	0		The number of this feed. Counts from 0 for each scan.
feedxoff	double	0.0		Beam offset for the cross-elevation axis, in degrees
feedeoff	double	0.0		Beam offset for the elevation axis, in degrees
sampler_name	string			The name of the sampler (a GBT-specific term)
bandwidth	double	0.0		Total bandwidth in Hz
observed_frequency	double	0.0		The observed (sky) frequency in Hz at the center of the bandpass.
sideband	string			The sideband (U or L)

equinox	double	2000.0		The equinox, in years, of the longitude and latitude axis values, when appropriate.
radesys	string	FK5		The equitorial coordinate system when appropriate, e.g. FK5, FK4, GAPPT.
target_longitude	double	0.0	trgtlong	The target (source) longitude pointing direction in degrees in the same coordinate system as longitude_axis. From the GO FITS file.
target_latitude	double	0.0	trgtlat	The target (source) latitude pointing direction in degrees in the same coordinate system as latitude_axis. From the GO FITS file.
timestamp	string	default	timestamp	The timestamp given to the scan when it was taken. YYYY_MM_DD_HH:MM:SS. This can be used in data selection when there are repeated scan numbers.
date	pointer to string array	current date		Date (YYYY-MM-DD) (along with utc) corresponding to mjd at each integration
utc	pointer to double array	current time		UTC seconds since start of date. Corresponds to UTC of mjd array
mjd	pointer to double array	from current date and time		Modified Julian Date at mid-point of integration in days
longitude_axis	pointer to double array	0.0		The longitude pointing direction at each integration, in degrees in coordinate_mode at equinox
latitude_axis	pointer to double array	0.0		The latitude pointing direction at each integration, in degrees in coordinate_mode at equinox
lst	pointer to double array	from current time		LST seconds corresponding to mjd array and site_location.
azimuth	pointer to double array	0.0		The azimuth for each integration in degrees.
elevation	pointer to double array	0.0		The elevation for each integration in degrees.
subref_state	pointer to integer array	1		Subreflector state when subreflector nodding; 0=moving, 1=first position, -1=second position

In this table, the *Index* column shows the name that each field is known by in the index file. These are the names returned by the **listcols** command. If the *Index* entry is blank, that field is not represented in the index file and is not available for selection.

E More about Flagging Data

This section provides more information and examples about flagging.

When data requires flagging, an iterative approach to reduction is often useful. Here is one approach:

1. Calibrate the raw data.
2. Examine the calibrated data and determine whether any flagging is required to improve calibration.
3. If necessary, flag the offending data and return to step 1.
4. Write a new SDFITS file with calibrated data. In general, the new SDFITS file should contain an entry for each integration that will be considered as a candidate for the average.
5. When all data are calibrated and written to disk, specify the calibrated data file as the new source of input.
6. Again examine the data and use the flagging procedures to mark residual bad data to exclude from the average.
7. Average the data.
8. Examine the average and, if necessary, return to step 1 or step 5 and modify the flagging commands as necessary.
9. Proceed with analysis of the averaged spectrum.

Because of the iterative nature of the process, it is common to set and then unset flagging commands for a given data set. It is important to emphasize that *blanked data are not recoverable* without going back to data retrieval, but *flagged data are recoverable*. Flagging (setting flag rules) allows you to iteratively decide which data should be blanked during processing.

Data can be flagged either by specifying scan number, integration number, polarization number, IF number, feed number, and channel number, or by specifying the record number (location within a file) and channel number. It is permissible to mix these two methods in a single flag file, if desired. The data I/O system in GBTIDL applies the flags, blanking data as appropriate (some control over which flags are applied is possible, as described later in this document). Averaging, analysis, and display procedures in GBTIDL take the appropriate action when blanked data are encountered.

Flagging is intended mainly for uncalibrated and pre-averaged data. However, it is not forbidden to flag calibrated, averaged data. Use caution in such cases because the header parameters used in the parametrization of flags can be changed during averaging operations. For this reason, when flagging averaged data it is generally best to flag by record number. Flagging by record number also offers a finer level of detail. The **select** procedure can be useful in conjunction with flagging by record number when the normal flag procedure isn't sufficient (this is described in more detail later in this section).

In the iterative flagging scheme outlined earlier in this section, flagging in Step 3 should be parametrized by scan, polarization, etc. while flagging in step 6 should be parametrized by record number.

E.1 Using Flags in GBTIDL

Flag rules (flags) can be set from the command line with the procedures **flag** and **flagrec**. These procedures generate entries in the flag file associated with the current SDFITS file. The **flag** procedure has the following syntax:

```
flag, scan, intnum=intnum, plnum=plnum, ifnum=ifnum, fdnum=fdnum,
     sampler=sampler, bchan=bchan, echan=echan, chans=chans,
     chanwidth=chanwidth, idstring=idstring, scanrange=scanrange, /keep
```

and the flagrec procedure has the following syntax:

```
flagrec, record, bchan=bchan, echan=echan, chans=chans, chanwidth=chanwidth,
     idstring=idstring, /keep
```

One uses *idstring* to associate with a rule an identifying string that is typically a reminder of the reason for the flag.

Examples:

The following example shows how to flag a channel range for a small number of scans and integrations. Note that either the *scan* parameter or *scanrange* keyword is required but both can not be used at the same time. For the other parameters, if they are not specified, "all" is assumed. So in the first example, all polarizations are flagged. Also, notice that the integration numbers specified are 1 AND 3, not 1 through 3. To select a range, use intnum=[1,2,3] or intnum=seq(1,3) (the first example specifies all of the integrations to be flagged as integers, the second generates that sequence of integers using the "seq" function).

```
flag, [18,19,20], intnum=[1,3], bchan=512, echan=514, idstring="RFI"
```

Equivalently, using the *scanrange* keyword:

```
flag, scanrange=[18,20], intnum=[1,3], bchan=512, echan=514, idstring="RFI"
```

To flag all channels for a given integration in one scan:

```
flag, 15, intnum=3, idstring="spectrometer glitch"
```

To flag all data for the given three scans:

```
flag, [101,105,107]
```

To flag a record in a processed data file (a *keep* file):

```
flagrec, 15, idstring="Glitch", /keep
```

To flag two channel ranges in a given scan you could do this:

```
flagrec, 16, bchan=0, echan=10, idstring="Two RFI Spikes"
flagrec, 16, bchan=100, echan=110, idstring="Two RFI Spikes"
```

or abbrieviate it like this:

```
flagrec, 16, bchan=[0,100], echan=[10,110], idstring="Two RFI Spikes"
```

The next example flags uses *chans* and *chanwidth* to flag the same channels:

```
flagrec, 16, chans=[5,105], chanwidth=11, idstring="Two RFI Spikes"
```

The select procedure can be used along with flagrec to provide even more flexible flagging. In this example, the "RR" polarization of IF number 3 for all data with the source name "Orion" is flagged in channels 500 to 520:

```
emptystack                                                    ; Clear the stack first
select, source='Orion', polarization='RR', ifnum=3            ; Populate the stack
flagrec, astack(), bchan=500, echan=520, idstring='RFI-Orion'
```

Note that there may be more than one flag associated with a given *idstring*. If *idstring* is not specified in the **flag** or **flagrec** calls, it defaults to the string "unspecified".

E.2 Using Flags in Data Retrieval and Averaging Procedures

Flags are applied by the data I/O subsystem when data are retrieved from disk. All of the data retrieval procedures in GBTIDL (including calibration procedures such as **getnod** and **getfs** that do data retrieval as part of their operation) use the I/O subsystem, so flags are applied whenever you get data from disk.

All of these procedures allow you to fine tune which flag rules are actually applied via the *useflag* and *skipflag* keywords. The default is to use */useflag*, meaning that all flag rules are applied. You can turn off all flagging by using */skipflag*. In that case, no data will be blanked by the data retrieval process. You can also apply or not apply some of the flags by referring to them by their *idstring*. You can not use both the *useflag* and *skipflag* keywords in the same call. Unlike **unflag**, the data retrieval commands do not allow you to skip or use flags based on their ID number - only the *idstring* can be used as an argument to these keywords.

Examples:

```
getnod, 15                              ; Apply all flags
getnod, 15, /skipflag                   ; Do not use any flags
getnod, 15, useflag="RFI"               ; Only use the "RFI" flag
getnod, 15, useflag=["RFI","wind"]      ; Use "RFI" and "wind" flags only
getnod, 15, skipflag="RFI"              ; Use all flags EXCEPT "RFI"
```

All of the standard procedures in GBTIDL that in turn use these procedures also have the *useflag* and *skipflag* keywords.

E.3 Listing Flags

Use **listflags** to list all of the flags for the current data file, or only those flags having a specific *idstring*. The default **listflags** output shows all flags in their entirety, but the format sometimes is difficult to read. Appending the */summary* keyword to **listflags** aligns the columns but in order to do that, it may truncate the information in a particular column and so not all information may be shown.

Examples:

```
listflags, 'RFI'            ; Shows the flag information associated with the 'RFI' idstring
listflags, /summary         ; Shows all flags with the information aligned by column
```

To list all of the unique *idstring* values in the flag file use the **listids** command.

Example flag lists:

If one executes the flagging command:

```
flag, [35,36,37], intnum=[1,3], bchan=512, echan=514, idstring="RFI"
```

the **listflags** output will look like this:

```
#ID, RECNUM, SCAN, INTNUM, PLNUM, IFNUM, FDNUM, BCHAN, ECHAN, IDSTRING
0 * 35:37 1,3 * * * 512 514 RFI
```

The first line of the output identifies the contents of each column. Most of these fields are self-explanatory. The first field is an ID number that is assigned dynamically and is simply the location of that flag rule in this list. The ID number can be used in the **unflag** procedure to remove a flag rule.

Flagging a few more scans, not in a nice sequence:

```
flag, [40,42,44,47,48,50,56], intnum=[1,3], bchan=512, echan=514, idstring="More RFI"
```

adds one new line to the listflags output:

```
#ID, RECNUM, SCAN, INTNUM, PLNUM, IFNUM, FDNUM, BCHAN, ECHAN, IDSTRING
0 * 35:37 1,3 * * * 512 514 RFI
1 * 40,42,44,47,48,50,56 1,3 * * * 512 514 More RFI
```

And **listflags,/summary** truncates the output and produces the following:

```
#ID, RECNUM, SCAN, INTNUM, PLNUM, IFNUM, FDNUM, BCHAN, ECHAN, IDSTRING
0 * 35:37 1,3 * * * 512 514 RFI
1 * 40,42,44,+ 1,3 * * * 512 514 More RFI
```

Notice how the scan information is truncated. Fields that contain more information than shown end in a plus sign, while asterisks indicate all values for that parameter are flagged (as in the unformatted **listflags** output).

The second column, RECNUM, is set when **flagrec** is used. For example:

```
flagrec, 15, bchan=0, echan=8, idstring="bad channels"
listflags

#ID, RECNUM, SCAN, INTNUM, PLNUM, IFNUM, FDNUM, BCHAN, ECHAN, IDSTRING
0 * 35:37 1,3 * * * 512 514 RFI
1 * 40,42,44,47,48,50,56 1,3 * * * 512 514 More RFI
2 15 * * * * * 0 8 bad channels
```

E.4 Undoing Flags

If you would like to remove all the flags associated with a given SDFITS file, you can simply remove the associated flag file and restart GBTIDL. Alternatively, flags can be unset using the **unflag** procedure. The **unflag** procedure takes a single parameter, *id*, and it removes all flagging commands that have that *id*, where *id* can either be a string matching an *idstring* value or an integer matching an ID number as shown by **listflags**.

```
unflag, id
```

If you want to re-flag that same data, you have to reissue the **flag** or **flagrec** commands. The *id* parameter can be either a scalar or an array, to unflag multiple entries at once.

Unflagging by ID number is simple and appealing but users should be familiar with the following very important feature. Since the ID number is generated dynamically, it changes after each flagging-related command, including the **unflag** command. Users should always use **listflags** before each use of **unflag** to be sure that they are using the appropriate ID value. Consider this example:

```
listflags

#ID, RECNUM, SCAN, INTNUM, PLNUM, IFNUM, FDNUM, BCHAN, ECHAN, IDSTRING
0 * 35:37 1,3 * * * 512 514 RFI
1 * 40,42,44,47,48,50,56 1,3 * * * 512 514 More RFI
2 15 * * * * * 0 8 bad channels
```

If you want to unflag the last 2 IDs, so you might (mistakenly) try the following:

```
unflag, 1
unflag, 2
% FLAGS::UNFLAG_ID: ID could not be found to unflag:        2
```

The error happens because the first unflag causes the remaining two flag rules to be renumbered to 0 and 1, and so there is no ID 2 to unflag any more. This would have been a more dangerous, silent error had there been more than 3 rules to begin with.

The correct way to unflag the entries:

```
listflags
unflag, 1
listflags
unflag, 1
```

or:

```
listflags
unflag, [1,2]
```

E.5 Weighting Issues not Addressed by this Flagging Scheme

You should be aware of some potential issues with the weights when averaging flagged data.

Consider two reduced spectra, A and B, which resulted from an average of flagged data. In each of the two spectra, the individual channels have been flagged to different extents, so the final noise in each channel differs depending on how much of the raw data were flagged going into the average. For example, channels 0-10 in A may have been heavily flagged prior to averaging, and so they contain a higher noise than the other channels in A. If the observer then wishes to average A and B, the weighting in the average will be wrong because relative weights have not been stored for these spectra on a channel-per-channel basis.

F Other GBTIDL Features and Examples

F.1 Customizing the output of the list procedure

The **list** command can be used to view detailed information on records in the active input file. Because this file contains a great deal of information, it is possible to request that the **list** command show only that information of interest.

The following command shows how to create the sample data used in these examples:

```
% sdfits -mode=raw -scans=177:180 -backends=acs /home/archive/test-data/tape-0002/TREG_040922
```

Then to access this data in GBTIDL use this command:

```
filein, 'TREG_040922.raw.acs.fits'
```

In the first example, we list all the records, or spectra, currently found in the index file.

```
GBTIDL -> list            ; Show a brief description of everything

#INDEX   SOURCE   SCAN   PROCEDURE   POL   IFNUM   FDNUM   INT   SIG   CAL
   0      W3OH    177      OffOn     XX      0       0      0     T     T
   1      W3OH    177      OffOn     XX      0       0      0     T     F
   2      W3OH    177      OffOn     XX      0       0      1     T     T
   3      W3OH    177      OffOn     XX      0       0      1     T     F
   4      W3OH    177      OffOn     YY      0       0      0     T     T
   .
   .
   .
   30     W3OH    180      OffOn     YY      0       0      1     T     T
   31     W3OH    180      OffOn     YY      0       0      1     T     F
```

Next, we use a simple search parameter:

```
GBTIDL -> list, index=[0,1,2]        ; Show a description of the first three records

#INDEX  SOURCE  SCAN  PROCEDURE  POL  IFNUM  FDNUM  INT  SIG  CAL
   0     W3OH   177     OffOn    XX     0      0     0    T    T
   1     W3OH   177     OffOn    XX     0      0     0    T    F
   2     W3OH   177     OffOn    XX     0      0     1    T    T
```

To see all of information associated with these records, use the **verbose** keyword.

```
GBTIDL -> list, index=[0,1,2], /verbose
```

This will return the parameters:

```
# INDEX, PROJECT, FILE, EXT, ROW, SOURCE, PROCEDURE, OBSID, E2ESC, PROCS, SCAN, POL,
PLNUM, IFNUM, FEED, FDNUM, INT, NUMCHN, SIG, CAL, SAMPLER, AZIMU, ELEV, LONGITUDE,
LATITUDE, TRGTLONG, TRGTLAT, LST, CENTFREQ, RESTFREQ, VELOCITY, FREQINT, FREQRES,
DATEOBS, TIMESTAMP, BANDWIDTH, EXPOSURE, TSYS, NSAVE
```

Obviously, the above example is not best if you are only interested in the values of a few specific columns. You can narrow the output like so:

```
GBTIDL -> list, index=[0,1,2], columns=["INDEX","INT","POLARIZATION"]

#INDEX  INT  POL
   0     0   XX
   1     0   XX
   2     1   XX
```

The **list** command prints records in the order of the index number by default. This can be changed using the *sortcol* keyword. Note that the full name of the column must be used.

```
GBTIDL -> list, scan=177, sortcol="INT"      ; Sort by integration number

#INDEX  SOURCE  SCAN  PROCEDURE  POL  IFNUM  FDNUM  INT  SIG  CAL
   0     W3OH    177    OffOn    XX     0      0     0    T    T
   1     W3OH    177    OffOn    XX     0      0     0    T    F
   4     W3OH    177    OffOn    YY     0      0     0    T    T
   5     W3OH    177    OffOn    YY     0      0     0    T    F
   2     W3OH    177    OffOn    XX     0      0     1    T    T
   3     W3OH    177    OffOn    XX     0      0     1    T    F
   6     W3OH    177    OffOn    YY     0      0     1    T    T
   7     W3OH    177    OffOn    YY     0      0     1    T    F
```

The **liststack** command is identical to the **list** command except that it selects from records identified by the stack.

```
GBTIDL -> select, scan=177        ; Place scan 177's records on the stack
GBTIDL -> liststack, col=["INDEX","INT","CAL"], sortcol="CAL"

#INDEX  INT  CAL
   1     0    F
   3     1    F
   5     0    F
   7     1    F
   0     0    T
   2     1    T
   4     0    T
   6     1    T
```

F.2 Making postage stamp plots

In displaying PointMap data, or just for displaying multiple spectra, it can be convenient to display spectra as postage stamp plots. GBTIDL currently does not have any inherent support for postage stamp plots, but it is easy to use the IDL plotter to duplicate plots as one might see from CLASS, for example.

For instance, a 3x3 PointMap might be stored as calibrated, reduced spectra in an SDFITS file, with the first 9 records representing the map. These can be displayed in a postage stamp plot as follows:

```
filein, 'postage.fits'
freeze
!p.multi = [0,3,3]
     for i=0,8 do begin & $
     getrec, i & $
     x = getxarray() & $
     y = getdata() & $
     plot, x, y, xstyle=1 & $
endfor
unfreeze
```

For more flexibility in plot placement, the **position** parameter can be used, as in the following procedure:

```
pro plotpos, x, y, xpos, ypos, xsize, ysize
     if (n_elements(xsize)) eq 0 then xsize = 0.1
     if (n_elements(ysize)) eq 0 then ysize = 0.1
     freeze
     plot,x,y,position=[xpos-xsize/2,ypos-ysize/2,xpos+xsize/2,ypos+ysize/2], $
          /noerase, xstyle=1
     unfreeze
end
```

The procedure might be used as follows:

```
erase
getrec,0
plotpos, getxarray(), getdata(), 0.5, 0.5, 0.25, 0.25
getrec,1
plotpos, getxarray(), getdata(), 0.2, 0.5, 0.25, 0.25
```

```
getrec,2
plotpos, getxarray(), getdata(), 0.8, 0.5, 0.25, 0.25
getrec,3
plotpos, getxarray(), getdata(), 0.5, 0.2, 0.25, 0.25
getrec,4
plotpos, getxarray(), getdata(), 0.5, 0.8, 0.25, 0.25
```

F.3 Example reduction sessions with sample data sets

This section describes a few sample data sets for users who may wish to experiment with GBTIDL but who do not yet have any data to play with. You may wish to experiment with GBTIDL before you have an appropriate data set of your own. With each data set is an example of how the data might be reduced and analyzed in GBTIDL. The examples are simply guides, and there are many ways to reduce the data in each case.

Example 1: HI Position Switched Data

This is a strightforward observation of HI in a galaxy, observed using position switching. The data set is "clean", so all the data can be included in the averaging. The example data reduction is terse in this case, and aims just to produce an HI spectrum calibrated as antenna temperature (K). The RMS noise and integrated flux density of the HI source are measured.

- Retrieve the data: ngc5291.fits
  (http://safe.nrao.edu/wiki/pub/GB/Data/GBTIDLExampleAndSampleData/ngc5291.fits)
- Example data reduction:

  Get the data into GBTIDL and show a summary of the scans:

  ```
  filein, 'ngc5291.fits'
  summary
  ```

  Calibrate and accumulate the data for each scan, and for each polarization:

  ```
  for i=51,57,2 do begin getps, i, plnum=0 & accum & end
  for i=51,57,2 do begin getps, i, plnum=1 & accum & end
  ave
  ```

  Set a baseline region and subtract the baseline:

  ```
  chan
  nregion,[3300,14800,17900,31000]
  nfit,3
  sety, 0.2, 0.5
  bshape
  baseline
  unzoom
  ```

  Apply some smoothing, then measure statistics:

  ```
  hanning,/decimate
  bdrop, 2500
  edrop, 2500
  velo
  stats, 2000, 3000       ; this gives the RMS: 13.5 mJy
  stats, 3900, 4800       ; this gives the integrated area: 60.439 K km/s
  boxcar, 8               ; more smoothing
  ```

Example 2: OH/HI Frequency Switched Data

This is a slightly more involved data set than the previous one. In this case, there are 2 spectral windows, or "IFs". One records the 1665/1667 MHz OH masers and the other records the HI emission toward W3(OH). The data are frequency switched. This data includes some integrations in which there are bad data, and so the observer must be careful to inspect and average the data selectively.

- Retrieve the data: W3OH.fits (http://safe.nrao.edu/wiki/pub/GB/Data/GBTIDLExampleAndSampleData/W3OH.fits)
- Example data reduction:

  Get the data into GBTIDL and show a summary of the scans:

```
filein, 'W3OH.fits'
summary
```

  Begin by visually inspecting the data. Note not all the data is "good" so we will need to be selective in the averaging. The "wait" command simply pauses to give the observer a chance to look at the data.

```
for i=79, 83 do begin getfs, i, plnum=1, ifnum=0 & wait, 2 & end
```

  Zoom in to the baseline and repeat:

```
sety, -2, 2
for i=79, 83 do begin getfs, i, plnum=1, ifnum=0 & wait, 2 & end
```

  Inspect individual integrations within scan 83. Note that within a scan some integrations are good and some bad.

```
for i=0,5 do begin getfs, 83,intnum=i, plnum=1, ifnum=0 & wait, 2 & end
```

  We must average only the good integrations. There are many ways to approach this problem. It would be natural to use the flagging commands, but here we use a different method which is terse but efficient. We loop through each integration of each scan, test the RMS in the data, and accumulate only the good integrations. The use of **freeze** before the loop and **unfreeze** after the loop speeds up the processing by turning off the automatic update of the plotter after each **getfs** call.

```
velo
freeze
for i=79,83 do begin & $
for j=0,5   do begin & $
for k=0,1   do begin & $
getfs, i, units='Jy', intnum=j, plnum=k, ifnum=0 & $
stats,-3000,-2000,ret=a,/quiet & $
if a.rms lt 0.5 then accum else print, 'Skipping' ,i, j, k & $
end & end & end
unfreeze
ave
```

  The next example illustrates the flagging approach. The bad integrations are first flagged and then the scans for ifnum=0 are averaged, using both polarizations. Note that the loop over integrations can now be eliminated. The **getfs** command averages all integrations and since the bad integrations are now flagged, they do not contribute to the average.

```
flag,[80,82], intnum=[1,3], plnum=1, ifnum=0, idstring='corrupt'
flag, 83, intnum=[2,4], plnum=1, ifnum=0, idstring='corrupt'
listflags,/summary
freeze
for i=79,83 do begin & $
for k=0,1 do begin  & $
getfs,i,units='Jy', plnum=k, ifnum=0 & $
accum & $
end & end
unfreeze
ave
```

Extract a region of interest:

```
chan
my_spec = dcextract(!g.s[0],7500,9500)
bdrop, 0
edrop, 0
show,my_spec
!g.s[0] = my_spec
show
```

Set the baseline regions using the mouse cursor and subtract a baseline.

```
sety, -0.2,0.4         ; Zoom in a bit
setregion
nfit, 7
bshape
baseline
```

Fit Gaussians to one of the maser complexes. Use **fitgauss** to specify a 3-component fit.

```
velo
setx, -60, -30
freey
fitgauss
```

Follow the instructions for **fitgauss**.

Example 3: H_2O Total Power Nod Data

This data set contains an observation of a maser line, observed in total power nod mode. In the first example below we show the simplest (but verbose) method to average and reduce the data. The second example is more involved. We use the stack to gather the scans for averaging. We store the individual scans in internal buffers, and display them all overlaid. Finally we average the data and write the final spectrum to disk.

- Retrieve the data: IC1481.fits
  (http://safe.nrao.edu/wiki/pub/GB/Data/GBTIDLExampleAndSampleData/IC1481.fits)
- Simple reduction of this data set:

  Get the data into GBTIDL and accumulate some of the data:

  ```
  filein, 'IC1481.fits'
  getnod, 182, plnum=0
  accum
  getnod, 182, plnum=1
  accum
  getnod, 184, plnum=0
  accum
  getnod, 184, plnum=1
  accum
  ```

The other scans can be accumulated similarly. Now, average the accumulated data and fit a baseline.

```
ave
setregion
nfit, 3
baseline
```

- Alternative reduction:

  Get the data into GBTIDL and show a summary of the scans:

  ```
  filein, 'IC1481.fits'
  summary
  ```

  Clear the stack, then fill it with even scan numbers in the range 182-188.

  ```
  emptystack
  sclear
  addstack, 182, 188, 2
  tellstack
  ```

  Now loop through each scan pair, retrieve the calibrated spectrum, accumulate it and also store it in a memory buffer. The use of **freeze** and **unfreeze** before and after the loop speeds up the processing by disabling the automatic update of the plotter after each getnod.

  ```
  freeze
  for i = 0, !g.acount-1 do begin & $
  getnod, astack(i), plnum=0, units='Jy', tsys=60 & accum & $
  copy, 0, i*2+2 & $
  getnod, astack(i), plnum=1, units='Jy', tsys=60 & accum & $
  copy, 0, i*2+3 & $
  end
  unfreeze
  ave
  ```

  Fit a baseline:

  ```
  setregion
  nfit, 3
  bshape
  baseline
  ```

  Smooth the spectrum, then save it to disk.

  ```
  hanning, /decimate
  fileout, 'saved.fits'
  keep
  ```

  Create a plot showing each individual spectrum (2 polarizations per scan pair) on a single plot, with offsets to make it easier to see the spectra:

  ```
  copy, 2, 0
  baseline
  show
  copy, 0, 2
  freeze
  for i=3,9 do begin copy, i, 0 & baseline & bias, float(i-2)*0.2 & copy, 0, i & end
  show, 2
  unfreeze
  for i=3,9 do oshow, i, color=!red
  ```

G Reducing Continuum Data

Limited processing of continuum data is available in GBTIDL. The **cont** procedure is used to switch to continuum mode. Continuum data comes from a separate file (also opened with **filein**) and there is a completely separate set of data containers in `!g` for holding continuum data (`!g.c`) . Continuum data can only be displayed with the x-axis as sample number (you do not need to do anything other than **show** to make that happen). There is currently no ability to save continuum data containers to disk, so the continuum functionality is rather limited. Many routines simply refuse to work with continuum data (e.g. getfs). However, the data is available in GBTIDL and so some work can be done with continuum data. Use the **line** procedure to switch back to spectral-line mode.

H More Information

H.1 Contributing Procedures

GBTIDL is intended to be built and expanded by the user base – that's you! To contribute a procedure to GBTIDL, send it by email to Jim Braatz at jbraatz@nrao.edu.

Visit the Contributed Code Reference (http://wwwlocal.gb.nrao.edu/GBT/DA/gbtidl/release/contrib/index.html) to see what user contributions are available.

Before you contribute a procedure it should be thoroughly tested. Be sure it includes parameter checks. Please document the procedure in comment lines at the top of the file, include examples of how to use it (if appropriate), and include your name and email address if you'd like credit for your hard work.

H.2 Bugs and Enhancements

Visit the bug submission page at http://gbtidl.nrao.edu on the GBTIDL home page. You need not be registered on Sourceforge to submit a bug or enhancement request. Past submissions can be tracked from the Sourceforge pages.

H.3 General Hints and Tips

H.3.1 Calibrating Data

One strategy in using GBTIDL is to first calibrate your data using a procedure in the (getfs, getnod, getoffon, ...) family, and then write the calibrated data to a new SDFITS file. Once the data are in the new file, the calibrated scans can be retrieved with the **getrec** procedure, and tools such as the stack and **select** and **find** can be used to access data groupings.

H.3.2 Recovering from Errors

In GBTIDL, when an error is encountered you are sometimes left at the "procedure level" and not the "main level" of the command line interface. GBTIDL will not work at the procedure level. To return to the main level, use the command "retall" and GBTIDL will resume. IDL users, in general, type "retall" often. This should only happen rarely in GBTIDL. If you find that it happening unexpectedly or frequently, please let us know.

H.4 GBTIDL FAQ

H.4.1 Do I need to know IDL to run GBTIDL?

No, but it helps. The GBTIDL syntax is similar to UniPOPs and it is possible to reduce many types of observations without much previous IDL knowledge. The GBTIDL User's Guide shows you how. However, if you'd like to go beyond the standard reduction facilities provided, some IDL knowledge is required. If you are not familiar with IDL and would like to learn, you may find the IDL primer useful. Links to a few primers can be found in section 1.6.

H.4.2 What is the latest version of GBTIDL?

Version 2.8 was released on June 1, 2011. It is available as **gbtidl** for all users in Green Bank, Charlottesville and Socorro. It can be downloaded from the GBTIDL home page.

H.4.3 What version of IDL is required to run GBTIDL?

GBTIDL runs on version 6.1 or later. It does not run on earlier version of IDL.

H.4.4 Everything was working fine, then I encountered an error and now GBTIDL is not working. How do I recover?

Try "retall". The error handling mechanism in IDL leaves you at the "procedure level" after an error is encountered. GBTIDL runs at the "main level". To return to the main level, use the IDL command "retall".

H.4.5 The plotter is not responding, how do I recover?

Try "retall". If you can't change the x-axis units, or print, or otherwise interact with the plotter graphically then the most likely explanation is that there was an error at the "procedure level" and IDL is not at the "main level". IDL only processes these GUI events when it is at the "main level" so when an error occurs, the plotter appears unresponsive.

H.4.6 I have a collection of IDL procedures. Where should I put them so that I can use them in GBTIDL?

The current directory is always searched first by IDL for files to compile. If you always run GBTIDL from the same directory, it is simplest to put your .pro files there. If you run GBTIDL from different directories and you want to collect your .pro files all in one place, place them in $HOME/gbtidlpro. GBTIDL includes $HOME/gbtidlpro at the head of the search path (after the current directory). So, any files you put in $HOME/gbtidlpro and in any subdirectories under $HOME/gbtidlpro will be found - even if a duplicate named file exists in the GBTIDL installation. $HOME/gbtidlpro could be a symbolic link to any other location.

H.4.7 How do I change the Y-axis label?

There are a few ways:

1. When you get the scan, use the unit specifier. This will automatically scale the spectra to the new units.

```
getfs, 79, units='Jy'
show                      ; If auto update is off
```

2. Set the units field in the data container:

```
!g.s[0].units = 'Flux Density (mJy)'
show
```

3. Use the predefined units :

```
If !g.s[0].units is 'Jy'  then label is 'Flux Density (Jy)'
                    'Ta*'               'Antenna Temperature (Ta*)'
                    'Ta'                'Antenna Temperature (Ta)'
                     empty              'Intensity'
                    'any other string'  'any other string'
```

4. This technique is not recommended because the string is not saved with the data (it only affects the current contents of the plotter), but this will also change the y-axis label:

```
show
(getplotterdc()).units = 'Flux Density (mJy)'
reshow
```

H.4.8 Can I use GBTIDL with data from telescopes other than the GBT?

If the data are in UniPOPS format you can use the **uni2sdfits** procedure (see details at the end of this answer). Otherwise, GBTIDL provides no tools for converting data from other telescopes. However, with a little work on your part, it should be possible to get spectral line data from any other telescope into GBTIDL. There is even some chance that if your data follow the SDFITS convention that GBTIDL will be able to read it directly.

The easiest way to import generic data into GBTIDL is to use standard IDL tools to get the data into a data container, as described below. You can then use GBTIDL to operate on data in data containers, and you can save the data containers to SDFITS format as well.

A data container is just an IDL data structure with a predefined format. See the 'About Data Containers' section for a discussion about data containers in GBTIDL.

You can get data into a data container as follows:

- Read your data values into an IDL array (e.g. use **READU** to read unformatted binary values or **mrdfits** to read FITS tables, or import the data from an ASCII file using readf)
- Create a new data container to hold those values: `dc = data_new(myvalues)`
- Set the associated header fields in `dc`, for example:

  ```
  dc.source = 'mysource'
  dc.scan = 24
  dc.coordinate_mode = 'J2000'
  ...
  ```

- Copy `dc` to the primary data container:

  ```
  set_data_container, dc
  ```

- Save it to an output file:

```
fileout,'myfile.fits'
keep
```

- Free the memory used by the data container:

```
data_free, dc
```

- When you return to look at this data in a later session, you can load it into GBTIDL directly from the SDFITS file as follows:

  start a new GBTIDL session first

```
filein,'myfile.fits'
getrec, 0
```

Pointers:

- When setting the header fields in dc, pay particular attention to those related to the frequency axis and its conversion to velocity.
- If you are going to be using set_data_container repeatedly, use freeze to speed up the operation.

Please contact us with any questions about the contents of the data container or other aspects of GBTIDL necessary to do this translation. (However, please be advised that we have limited resources, and will not be able to provide extensive development for reducing data from non-NRAO telescopes.)

As an example, the contributed procedure uni2sdfits uses an IDL class sdd to read in the UniPOPS binary file and then the above steps are used to copy it to the primary data container and keep it to the output file. The source code can be seen by clicking on the "source" links at the top of the page in each of the above two links.

H.5 Who Developed GBTIDL?

Bob Garwood, Paul Marganian, Jim Braatz, Nicole Radziwill, and Ron Maddalena, all of NRAO. Significant contributions have come from many others, especially: Tom Bania (BU), Phil Jewell (NRAO), Frank Ghigo (NRAO), Glen Langston (NRAO), David Fleming (UIUC), and Tim Robishaw (Berkeley).

H.6 Installing GBTIDL on a Mac

The standard GBTIDL installation instructions should be followed. The NRAO does not have a standard Mac installation or centralized support for Macs and NRAO Macs do not have GBTIDL installed on them in a standard location. The following troubleshooting tips may be helpful in setting up GBTIDL on a Mac:

- Make sure that the complete path name of IDL is set early on in the "gbtidl" startup script. It should look something like this (replace `my_path_for_idl` with the appropriate path on your Mac):

```
LOCAL_IDL=/Applications/my_path_for_idl/idl
```

- Make sure "idl" itself is properly installed on your Mac.

- Make sure your path includes "idl". When you type "idl" at the command prompt, the ITT Visual Information Services program should start (RSI on older versions of "idl"), not some other Mac program (there is a code utility also called "idl" that appears on some Macs). If you do not see the expected "idl", then set your path appropriately.
- If you are running on an NRAO computer or some other system using shared licenses, make sure you have license access set up properly.
- You should set your PATH variable to include the gbtidl executable. Assuming you have the "gbtidl" script in the directory called `/Users/my_account/GBTIDL` then a line like this in your .bashrc file would accomplish that:

```
export PATH=$PATH:/Users/my_account/GBTIDL
```

Index