MCW *AFNI* — Buckets

Robert W. Cox, Ph.D.

rwcox@mcw.edu

The New 'Bucket' Dataset Type

A **bucket** dataset is a 3D dataset that can contain an arbitrary number of sub-bricks. These sub-bricks are not considered to be time-ordered; rather, the bucket dataset type is a place where the user can toss 3D bricks of data.

This documentation is provided to update the *AFNI* plugins manual and to explain the programmatic interface for creating bucket datasets. It is current as of *AFNI* 2.20, and is at present a work-in-progress (*i.e.*, subject to change at a moment's whim).

Note that the entire MCW *AFNI* package—including plugins—must be recompiled to use these features. This is because the internal storage scheme used for datasets has been modified slightly.

Contents

It has always been possible to create a 3D (no time) dataset with multiple sub-bricks. Until now, there has not been any program that would do this, nor would any but the first sub-brick be visible from within *AFNI*. The new program `3dbucket` allows the creation of datasets with an arbitrary number of sub-bricks drawn from the bricks of existing dataset. *AFNI* has been modified to allow the user to switch viewing between sub-bricks. The dataset structure has been extended to allow extra information to be attached to each sub-brick to make them easy to distinguish. A programming interface has been implemented that allows external programs (and plugins) to create 3D bucket datasets.

1 Sub-brick Auxiliary Data

Three new types of data can be associated with each sub-brick in any *AFNI* dataset (bucket, 3D+time, ...). They are

Label This is a character string that is displayed on the *AFNI* bucket chooser menu that lets the user decide which sub-brick should be displayed (see §5).

Keywords This is a character string that contains a list of keywords that are to be associated with a given sub-brick. Each keyword is separated by the C substring `" ; "`. At present, the keywords have no function within any MCW *AFNI* program, but that is likely to change shortly.

Statistical Parameters Each sub-brick can have a statistical type attached, exactly as some of the earlier function types can. If a sub-brick with a valid statistical type is chosen to be the threshold sub-brick from within *AFNI*, then the nominal p-value per voxel will be displayed beneath the threshold slider.

Most of these sub-brick statistical types require auxiliary parameters. The list of statistical types is:

Name	Type Code	Distribution	Auxiliary parameters
`fico`	`FUNC_COR_TYPE`	Correlation Coeff	# Samples, # Fit Param, # Ort Param
`fitt`	`FUNC_TT_TYPE`	Student t	Degrees-of-Freedom (DOF)
`fift`	`FUNC_FT_TYPE`	F ratio	Numerator DOF, Denominator DOF
`fizt`	`FUNC_ZT_TYPE`	Standard Normal	— none —
`fict`	`FUNC_CT_TYPE`	Chi-Squared	DOF
`fibt`	`FUNC_BT_TYPE`	Incomplete Beta	Parameters a and b
`fibn`	`FUNC_BN_TYPE`	Binomial	# Trials, Probability per trial
`figt`	`FUNC_GT_TYPE`	Gamma	Shape, Scale
`fipt`	`FUNC_PT_TYPE`	Poisson	Mean

The 'Name' is used on the command line when modifying the auxiliary data inside a dataset using the program `3drefit`. The 'Type Code' is a C macro for a constant that is used from within a program when modifying the auxiliary data.

In addition to the sub-brick specific keywords list, I have also added a global keywords list that pertains to the entire dataset. One ultimate purpose of the keywords lists is to allow the selection of datasets and sub-bricks based on keywords.

It is possible to attach a label and a statistical type to sub-bricks of non-bucket datasets. But they will have no effect.

2 Program `3dbucket`

At this moment, the only program that can create a bucket dataset is `3dbucket`. (In particular, `to3d` *cannot* create a bucket dataset!) `3dbucket` concatenates 3D sub-bricks from multiple input datasets and produce one output bucket dataset. The main purpose of this program is to experiment with bucket datasets. Its help file follows:

```
Usage: 3dbucket options
where the options are:
     -prefix pname = Use 'pname' for the output dataset prefix name.
 OR  -output pname     [default='buck']

     -session dir  = Use 'dir' for the output dataset session directory.
                       [default='./'=current working directory]
     -dry          = Execute a 'dry run'; that is, only print out
                       what would be done.  This is useful when
                       combining sub-bricks from multiple inputs.
     -verb         = Print out some verbose output as the program
                       proceeds (-dry implies -verb).
     -fbuc         = Create a functional bucket.
     -abuc         = Create an anatomical bucket.  If neither of
                       these options is given, the output type is
                       determined from the first input type.

Other arguments are taken as input datasets.  A dataset is specified
using one of the forms
   'prefix+view', 'prefix+view.HEAD', or 'prefix+view.BRIK'.
You can also add a sub-brick selection list after the end of the
dataset name.  This allows only a subset of the sub-bricks to be
included into the output (by default, all of the input dataset
is copied into the output).  A sub-brick selection list looks like
one of the following forms:
  fred+orig[5]                     ==> use only sub-brick #5
  fred+orig[5,9,17]                ==> use #5, #9, and #12
  fred+orig[5..8]     or [5-8]     ==> use #5, #6, #7, and #8
  fred+orig[5..13(2)] or [5-13(2)] ==> use #5, #7, #9, #11, and #13
Sub-brick indexes start at 0.  You can use the character '$'
```

```
to indicate the last sub-brick in a dataset; for example, you
can select every third sub-brick by using the selection list
  fred+orig[0..$(3)]

N.B.: The sub-bricks are output in the order specified, which may
 not be the order in the original datasets.  For example, using
  fred+orig[0..$(2),1..$(2)]
 will cause the sub-bricks in fred+orig to be output into the
 new dataset in an interleaved fashion.  Using
  fred+orig[$..0]
 will reverse the order of the sub-bricks in the output.

N.B.: Bucket datasets have multiple sub-bricks, but do NOT have
 a time dimension.  You can input sub-bricks from a 3D+time dataset
 into a bucket dataset.  You can use the '3dinfo' program to see
 how many sub-bricks a 3D+time or a bucket dataset contains.

N.B.: The '$', '(', ')', '[', and ']' characters are special to
 the shell, so you will have to escape them.  This is most easily
 done by putting the entire dataset plus selection list inside
 single quotes, as in 'fred+orig[5..7,9]'.
```

Some additional points:

- If an input sub-brick has a statistical type, then its type and auxiliary parameters are copied to the output sub-brick. This happen if the input dataset is one of the functional types with a statistics threshold attached (*e.g.*, the second sub-brick from a `fico` dataset). It can also happen if the input dataset is itself a bucket dataset.
- The sub-brick labels for the output dataset are of the form `prefix[index]`, where '`prefix`' is the input dataset and '`index`' is the integer index of the sub-brick in the input dataset.
- The output sub-brick keywords are copied from the input sub-bricks, if any. The additional keyword `prefix+view[index]` is also attached to the sub-brick keyword list.
- I intend to extend the input sub-brick selection scheme for `3dbucket` to allow selection from keyword lists. Eventually, it will also be possible to construct datasets 'on-the-fly' on the command line for any program.
- Anatomical bucket datasets are not particularly useful, at least at present. (Got any ideas for applications?)

3 Program 3drefit

This program lets the user change the contents of a dataset header. It has been extended to let the sub-brick auxiliary data be modified. Its help file follows:

```
Usage: 3drefit [options] dataset ...
where the options are
  -orient code    Sets the orientation of the 3D volume(s) in the .BRIK.
                  The code must be 3 letters, one each from the
                  pairs {R,L} {A,P} {I,S}.  The first letter gives
                  the orientation of the x-axis, the second the
                  orientation of the y-axis, the third the z-axis:
                     R = right-to-left         L = left-to-right
                     A = anterior-to-posterior P = posterior-to-anterior
                     I = inferior-to-superior  S = superior-to-inferior
               ** WARNING: when changing the orientation, you must be sure
                  to check the origins as well, to make sure that the volume
                  is positioned correctly in space.

  -xorigin distx  Puts the center of the edge voxel off at the given
  -yorigin disty  distance, for the given axis (x,y,z); distances in mm.
  -zorigin distz  (x=first axis, y=second axis, z=third axis).
                  Usually, only -zorigin makes sense.  Note that this
                  distance is in the direction given by the corresponding
                  letter in the -orient code.  For example, '-orient RAI'
                  would mean that '-zorigin 30' sets the center of the
                  first slice at 30 mm Inferior.  See the to3d manual
                  for more explanations of axes origins.
               ** SPECIAL CASE: you can use the string 'cen' in place of
                  a distance to force that axis to be re-centered.

  -xdel dimx      Makes the size of the voxel the given dimension,
  -ydel dimy      for the given axis (x,y,z); dimensions in mm.
  -zdel dimz   ** WARNING: if you change a voxel dimension, you will
                  probably have to change the origin as well.

  -TR time        Changes the TR time to a new value (see 'to3d -help').
               ** WARNING: this only applies to 3D+time datasets.

  -newid          Changes the ID code of this dataset as well.

  -statpar v ...  Changes the statistical parameters stored in this
                  dataset.  See 'to3d -help' for more details.
```

```
  -markers        Adds an empty set of AC-PC markers to the dataset,
                  if it can handle them (is anatomical, doesn't already
                  have markers, is in the +orig view, and isn't 3D+time).

  -appkey ll      Appends the string 'll' to the keyword list for the
                  whole dataset.
  -repkey ll      Replaces the keyword list for the dataset with the
                  string 'll'.
  -empkey         Destroys the keyword list for the dataset.

  -type           Changes the type of data that is declared for this
                  dataset, where 'type' is chosen from the following:
       ANATOMICAL TYPES
         spgr == Spoiled GRASS             fse == Fast Spin Echo
         epan == Echo Planar              anat == MRI Anatomy
           ct == CT Scan                  spct == SPECT Anatomy
          pet == PET Anatomy               mra == MR Angiography
         bmap == B-field Map              diff == Diffusion Map
         omri == Other MRI                abuc == Anat Bucket
       FUNCTIONAL TYPES
          fim == Intensity                fith == Inten+Thr
         fico == Inten+Cor                fitt == Inten+Ttest
         fift == Inten+Ftest              fizt == Inten+Ztest
         fict == Inten+ChiSq              fibt == Inten+Beta
         fibn == Inten+Binom              figt == Inten+Gamma
         fipt == Inten+Poisson            fbuc == Func-Bucket

The options below allow you to attach auxiliary data to sub-bricks
in the dataset.  Each option may be used more than once so that
multiple sub-bricks can be modified in a single run of 3drefit.

  -sublabel  n ll  Attach to sub-brick #n the label string 'll'.
  -subappkey n ll  Add to sub-brick #n the keyword string 'll'.
  -subrepkey n ll  Replace sub-brick #n's keyword string with 'll'.
  -subempkey n     Empty out sub-brick #n' keyword string

  -substatpar n type v ...
                  Attach to sub-brick #n the statistical type and
                  the auxiliary parameters given by values 'v ...',
                  where 'type' is one of the following:
```

```
type  Description  PARAMETERS
----  -----------  ---------------------------------------
fico  Cor          SAMPLES  FIT-PARAMETERS  ORT-PARAMETERS
fitt  Ttest        DEGREES-of-FREEDOM
fift  Ftest        NUMERATOR and DENOMINATOR DEGREES-of-FREEDOM
fizt  Ztest        N/A
fict  ChiSq        DEGREES-of-FREEDOM
fibt  Beta         A (numerator) and B (denominator)
fibn  Binom        NUMBER-of-TRIALS and PROBABILITY-per-TRIAL
figt  Gamma        SHAPE and SCALE
fipt  Poisson      MEAN
```

3drefit is the only program that lets the user change the sub-brick label, keywords, and statistical parameters. In particular, if you don't like the default labels provided by 3dbucket, you must use 3drefit to patch things up. Program 3dinfo has been modified so that it will print out the sub-brick auxiliary data (including keywords). This will help guide the use of 3drefit.

4 Creating Buckets in a Program

Modifying Sub-Brick Parameters

A number of new ADN_ commands have been added to EDIT_dset_items in order to make creating bucket datasets moderately painless. In combination with a couple of other utility routines, it is possible to create an empty dataset with n sub-bricks, attach data arrays to them and auxiliary data to them, and even later to expand the number of sub-bricks.

The new ADN_ commands are described below. (This section should be read in conjunction with §2.7 of the *AFNI* plugins manual.) The inputs to EDIT_dset_items are copied into the internal data structure of the dataset being modified, and so can be freed or otherwise recycled after this function return. Bucket construction examples will be given later.

Control Code	Data Type	Meaning
ADN_brick_label_one	char *	Unlike the earlier ADN_ codes, this one, and the others below that end in '_one' are be used to set auxiliary parameters for individual sub-bricks in a dataset. This is done by adding the sub-brick index to the ADN_ code. Note that only one version of any particular '_one' code can be used per call to EDIT_dset_items—to set multiple sub-bricks, a loop is required. This particular code is used to set the label that is displayed in the menu that is used to select which sub-brick is displayed.

ADN_brick_fac_one	float	This code is used to set the scale factor for an individual sub-brick. The alternative code, ADN_brick_fac (described in the plugins manual), is used to set *all* the sub-brick factors at once.
ADN_brick_stataux_one	float *	This code is used to set the auxiliary statistical parameters for an individual sub-brick. The float array that is passed as the paired argument must have the following contents: `statcode npar v1 v2 ... vn` where statcode is the type of statistic stored in the sub-brick, npar is the number of parameters that follow in the array, and v1 ... vn are the parameters for that type of statistic. (Note that npar may be 0.) See §1 for details on the different statistical types supported by *AFNI*.
ADN_brick_keywords_replace_one	char *	This code is used to delete the keywords associated with a sub-brick and replace them with a new set. The list of keywords is stored as a single string, with distinct entries separated by the substring " ; ". If you want to enter multiple distinct keywords with this operation, you must supply the " ; " yourself within the paired argument.
ADN_brick_keywords_append_one	char *	This code is used to add keywords to the list associated with a sub-brick. The input character string will be appended to the existing keyword string, with " ; " separating them. (This function will supply the " ; " separator.) If there are no keywords, this operation is equivalent to the replace function above.
ADN_keywords_replace	char *	This is used to replace the keywords list associated with the entire dataset.
ADN_keywords_append	char *	This is used to append to the keyword list for the entire dataset.

To make some of these steps easier, the following C macros have been defined:

- `EDIT_BRICK_LABEL(ds,iv,str)`
  will change the label in the ivth sub-brick of dataset ds to the string str.

- `EDIT_BRICK_FACTOR(ds,iv,fac)`
  will change the scaling factor in the ivth sub-brick of dataset ds to the float value fac.

- `EDIT_BRICK_ADDKEY(ds,iv,str)`
  will add the keyword string str to the ivth sub-brick of dataset ds.

- `EDIT_BRICK_TO_FICO(ds,iv,nsam,nfit,nort)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fico` type (correlation coefficient) with statistical parameters `nsam`, `nfit`, and `nort`.

- `EDIT_BRICK_TO_FITT(ds,iv,ndof)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fitt` (t-test) type with statistical parameter `ndof`.

- `EDIT_BRICK_TO_FIFT(ds,iv,ndof,ddof)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fift` (F-test) type with statistical parameters `ndof` and `ddof`.

- `EDIT_BRICK_TO_FIZT(ds,iv)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fizt` type (z-score, or normally distributed).

- `EDIT_BRICK_TO_FICT(ds,iv,ndof)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fict` type (χ^2 distributed) with statistical parameter `ndof`.

- `EDIT_BRICK_TO_FIBT(ds,iv,a,b)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fibt` type (beta distributed) with statistical parameters `a` and `b`.

- `EDIT_BRICK_TO_FIBN(ds,iv,ntrial,prob)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fibn` type (binomial distributed) with statistical parameters `ntrial` and `prob`.

- `EDIT_BRICK_TO_FIGT(ds,iv,shape,scale)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fign` type (gamma distributed) with statistical parameters `shape` and `scale`.

- `EDIT_BRICK_TO_FIPT(ds,iv,mean)`
  changes the `iv`$^{\text{th}}$ sub-brick of dataset `ds` to be `fipt` type (Poisson distributed) with statistical parameter `mean`.

Example: Creating a Bucket Dataset All at Once

In this example, an empty copy of an input dataset is made (to get the geometry correct), then the new dataset is turned into a function bucket, then the sub-bricks are attached. The following code is adapted from `3dbucket.c`.

```
THD_3dim_dataset * old_dset , * new_dset ;
char * output_prefix , output_session ;
int new_nvals , iv ;
short ** bar ;          /* bar[iv] points to data for sub-brick #iv */
char ** new_label ;     /* new_label[iv] points to label for #iv    */
char ** new_keyw ;      /* new_keyw[iv] points to keywords for #iv  */
float * new_fac ;       /* new_fac[iv] is new scale factor for #iv  */
float   sax[32] ;       /* statistical auxiliary parameters         */

/*-- Copy the input dataset structure, but not data --*/

new_dset = EDIT_empty_copy( old_dset ) ;

/*-- Modify some structural properties.
     Note that the new_nvals just makes empty
     sub-bricks, without any data attached.   --*/

EDIT_dset_items( new_dset ,
                   ADN_prefix        , output_prefix ,
                   ADN_directory_name, output_session ,
                   ADN_type          , HEAD_FUNC_TYPE ,
                   ADN_func_type     , FUNC_BUCK_TYPE,
                   ADN_ntt           , 0 ,              /* no time! */
                   ADN_nvals         , new_nvals ,
                   ADN_malloc_type   , DATABLOCK_MEM_MALLOC ,
                 ADN_none ) ;

if( THD_is_file(DSET_HEADNAME(new_dset)) ){
   fprintf(stderr,"*** Fatal error: file %s already exists!\n",
           DSET_HEADNAME(new_dset) ) ;
   exit(1) ;
}

/*-- Loop over new sub-brick index,
     attach data array with EDIT_substitute_brick
     (this just attaches the pointer, it DOES NOT copy the array),
     then put some strings into the labels and keywords,
     and modify the sub-brick scaling factor
     (a zero scaling factor means don't scale the data array). --*/
```

```
for( iv=0 ; iv < new_nvals ; iv++ ){
   EDIT_substitute_brick( new_dset , iv ,         /* attach bar[iv] to   */
                          MRI_short , bar[iv] ) ; /* be sub-brick #iv.   */
                                                  /* don't free bar[iv]! */

   EDIT_dset_items( new_dset ,
                       ADN_brick_label_one           +iv, new_label[iv] ,
                       ADN_brick_keywords_replace_one+iv, new_keyw[iv]  ,
                       ADN_brick_fac_one             +iv, new_fac[iv]   ,
                    ADN_none ) ;
}

/*-- Make sub-brick #2 be a t-test --*/

sax[0] = FUNC_TT_TYPE ;
sax[1] = 1.0 ;
sax[2] = degrees_of_freedom ;
EDIT_dset_items( new_dset ,
                   ADN_brick_stataux_one + 2 , sax ,
                 ADN_none ) ;

/*-- write new dataset to disk --*/

DSET_write( new_dset ) ;
```

Adding Sub-Bricks to a Bucket Dataset

In the above example, all the sub-bricks were created at once. They were initially empty, after the first call to `EDIT_dset_items`, but otherwise had all the structure needed. After a sub-brick has an actual data array attached to it, the `ADN_nvals` code can no longer be used to change the number of sub-bricks in dataset.

If a dataset already has actual data attached to any of its sub-bricks, another method must be used to add a new sub-brick:

```
short * qbar ;
float   qfac ;
EDIT_add_brick( new_dset , MRI_short , qfac , qbar ) ;
```

will create a new sub-brick in the dataset, with data type `short`, scale factor `qfac`, and data array `qbar`. (The pointer `qbar` is just copied into the sub-brick—the data it points to now 'belongs' to the dataset and should not be freed by you!) If you wish to attach a label, keywords, or statistical parameters to this new brick, you would do this using `EDIT_dset_items` (using the correct index for the new sub-brick).

Note that if you are doing this to a 3D+time dataset, as opposed to a bucket dataset, then a little more needs to be done. See `plug_realtime.c` for an example of how the *AFNI* real-time system uses `EDIT_add_brick` to grow a 3D+time dataset during image acquisition.

Accessing Sub-Brick Data

The following `C` macros can be used to access the contents of sub-bricks and their associated data. The argument `ds` is a pointer to a dataset `struct`, and the argument `iv` is a sub-brick index.

Macro	Meaning
`ISVALID_DSET(ds)`	Returns 1 if `ds` is a valid pointer to a dataset, or 0 if it is not.
`ISANATBUCKET(ds)`	Returns 1 if `ds` is an anatomy bucket dataset, 0 if it is not.
`ISFUNCBUCKET(ds)`	Returns 1 if `ds` is a function bucket dataset, 0 if it is not.
`ISBUCKET(ds)`	Returns 1 if `ds` is a bucket dataset (function or anatomy), 0 if it is not.
`DSET_BRICK_TYPE(ds,iv)`	Returns an integer describing what type of data is stored in the sub-brick array.
`DSET_BRICK_ARRAY(ds,iv)`	Returns a pointer to the sub-brick array.
`DSET_BRICK_FACTOR(ds,iv)`	Returns the sub-brick floating point scale factor.
`DSET_NVALS(ds)`	Returns the number of sub-bricks in a dataset.
`DSET_BRICK_LABEL(ds,iv)`	Returns a pointer to the sub-brick label. This pointer will not be `NULL`. Do *not* write into this string!
`DSET_BRICK_STATCODE(ds,iv)`	Returns an integer with the statistical type of a sub-brick. A positive value means that this sub-brick can be interpreted as a statistic. Note that if `ds` is on of the older 2-brick datasets such as `fico`, then calling this with `iv=1` will return the correct code, even though that code is actually associated with the dataset as a whole, not the sub-brick.
`DSET_BRICK_STATAUX(ds,iv)`	Returns a pointer to a `float` array with the statistical parameters for this sub-brick. This may be `NULL`, which means that you did something wrong. Do *not* write into this array! The number of parameters in this array can be determined from the table in §1, or from `FUNC_need_stat_aux[kv]` where `kv = DSET_BRICK_STATCODE(ds,iv)`.
`DSET_BRICK_STATPAR(ds,iv,jj)`	Returns the `jj`$^{\text{th}}$ statistical parameter for this sub-brick. This will be a float.

`DSET_BRICK_KEYWORDS(ds,iv)`	Returns a pointer to the keywords string for this sub-brick (`char *`). Do *not* write into this string! This pointer may be `NULL`.
`DSET_KEYWORDS(ds)`	Returns a pointer to the keywords string for the entire dataset. Do *not* write into this string! This pointer may be `NULL`.

Creating a Bucket from a 3D+time Dataset

I have written a utility routine to create a function bucket dataset from an input 3D+time dataset. This function takes as input a user-supplied function that returns the bucket values at each voxel. This function resides in `3dmaker.c` (a new file), and can be called from a plugin or from a command-line program. Its calling sequence is:

```
new_dset = MAKER_4D_to_typed_fbuc( THD_3dim_dataset * old_dset ,
                                   char * new_prefix , int new_datum ,
                                   int ignore , int detrend ,
                                   int nbrik ,generic_func * user_func ,
                                   void * user_data ) ;
```

The inputs to this function are:

`old_dset`	Pointer to old dataset; note that this dataset must not be warp-on-demand.
`new_prefix`	String to use as filename prefix.
`new_datum`	Type of data to store in output brick; if negative, will use datum from `old_dset`.
`ignore`	Number of data points to ignore at the start.
`detrend`	If nonzero, this routine will detrend (`a+b*t`) each time series before passing it to `user_func`.
`nbrik`	Number of values (and sub-bricks) to create at each voxel location.
`user_func`	Function to compute outputs; discussed below.
`user_data`	Discussed below.

The output is a pointer to a new dataset. If `NULL` is returned, some error occurred.

The `user_func` function should be declared like so:

```
void user_func( double tzero , double tdelta ,
                int npts , float ts[] , double ts_mean , double ts_slope ,
                void * ud , int nbrik , float * val ) ;
```

The arguments to `user_func` are:

`tzero`	Time at `ts[0]`.
`tdelta`	Time at `ts[1]` (*i.e.*, `ts[k]` is at `tzero+k*tdelta`); tzero and tdelta will be in sec if this is truly 'time'.

`npts`	Number of points in ts array.
`ts`	One voxel time series array, `ts[0]` ... `ts[npts-1]`; note that this will always be a float array, and that `ts` will start with the `ignore`$^{\text{th}}$ point of the actual voxel time series.
`ts_mean`	Mean value of `ts` array.
`ts_slope`	Slope of `ts` array; this will be inversely proportional to `tdelta` (units of 1/sec); if `detrend` is nonzero, then the mean and slope will been removed from the `ts` array.
`ud`	The `user_data` pointer passed in here; this can contain whatever control information the user wants.
`nbrik`	Number of output values that this function should return for the voxel corresponding to input data `ts`.
`val`	Pointer to return values for this voxel; note that this is a `float` array of length `nbrik`, and that values you don't fill in will be set to zero.

Before the first timeseries is passed, `user_func` will be called with arguments

```
( 0.0 , 0.0 , nvox , NULL , 0.0 , 0.0 , user_data , nbrik , NULL )
```

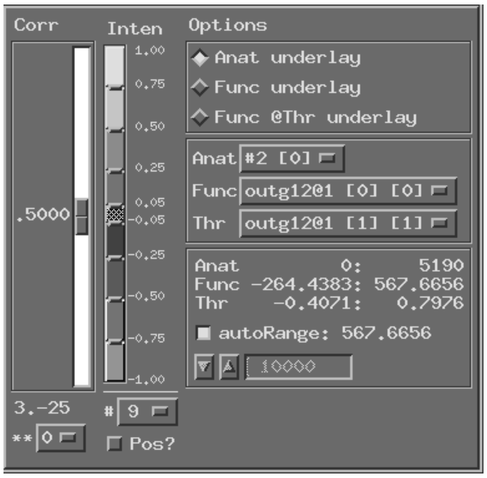
where `nvox` = number of voxels that will be processed. This is to allow for some setup (*e.g.*, `malloc`). After the last timeseries is passed, `user_func` will be called again with arguments

```
( 0.0 , 0.0 , 0 , NULL , 0.0 , 0.0 , user_data , nbrik , NULL )
```

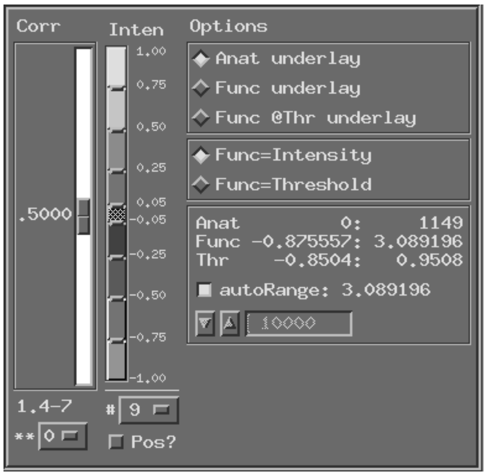
This is to allow for cleanup (*e.g.*, `free`). Note that the only difference between these 'notification' calls is the third argument. After the new dataset is created, you will likely want to modify some of the auxiliary data associated with its sub-bricks (*e.g.*, set statistical parameters and labels).

5 Changes in *AFNI*

If the active datasets are buckets, then the set of choosers in the `Define Function` control panel changes. The figure below shows the new bucket sub-brick choosers on the left; the old style sub-brick choosers are shown on the right for comparison (these are used with the non-bucket dataset types).



With bucket sub-brick choosers



With old-style choosers

Note that any sub-brick of a bucket dataset can be used as a threshold. This is true even if it does not have statistical parameters attached. (In that case, no *p*-value can be computed, of course.)

Two new buttons have been added to the $\boxed{\texttt{Misc}}$ menu under the `Define Datamode` control panel: $\boxed{\texttt{Anat Info}}$ and $\boxed{\texttt{Func Info}}$. These buttons will popup message windows with the output of program `3dinfo` about the current anatomical and functional datasets. This is to help look up keywords and statistical types of sub-bricks. In addition, the $\boxed{\text{FIM}}$ button has been removed from the `Define Function` control panel.

6 Still to Come

Things that are needed:

- Routines to deal with keyword lists, and a mechanism to allow the user to select sub-bricks (and datasets) based on keywords.
- A mechanism to allow the user to assemble datasets ‘on-the-fly’ using keyword and/or sub-brick index criteria. (A generalization of the syntax of `3dbucket` is a possibility.)
- Applications that create buckets (*e.g.*, multiple regression, …).