The DAVE 1.x data format

Document Objectives.

This document describes the internal data format used in DAVE. It also shows how to work with datasets written in the format within your applications. It does not contain information about attaching your application into the DAVE application suite – this is discussed elsewhere.

1. Description of format

DaVE 1.x uses a simple data structure to store experimental data. The structure was conceived early on in the development of DAVE and at the time emphasis was placed on ensuring that the *plottable data* is well represented. The complete data structure is created and saved on file as an IDL pointer (heap variable) hence it is commonly referred to as a ***davePtr*** (pronounced DAVE pointer). DAVE 1.x files are saved using IDL's sav binary format.

Essentially several data structures have been defined to store the relevant information within the DAVE pointer. The complete structure is outlined in the diagram below using the following rules.

- Each rectangular shape contains the definition of an IDL structure.
- Structure member names are in the left column (in blue).
- Their IDL data type and description are in the right column.
- Highlighted structures member names (bold-underlined) are themselves structures or pointers to structures.

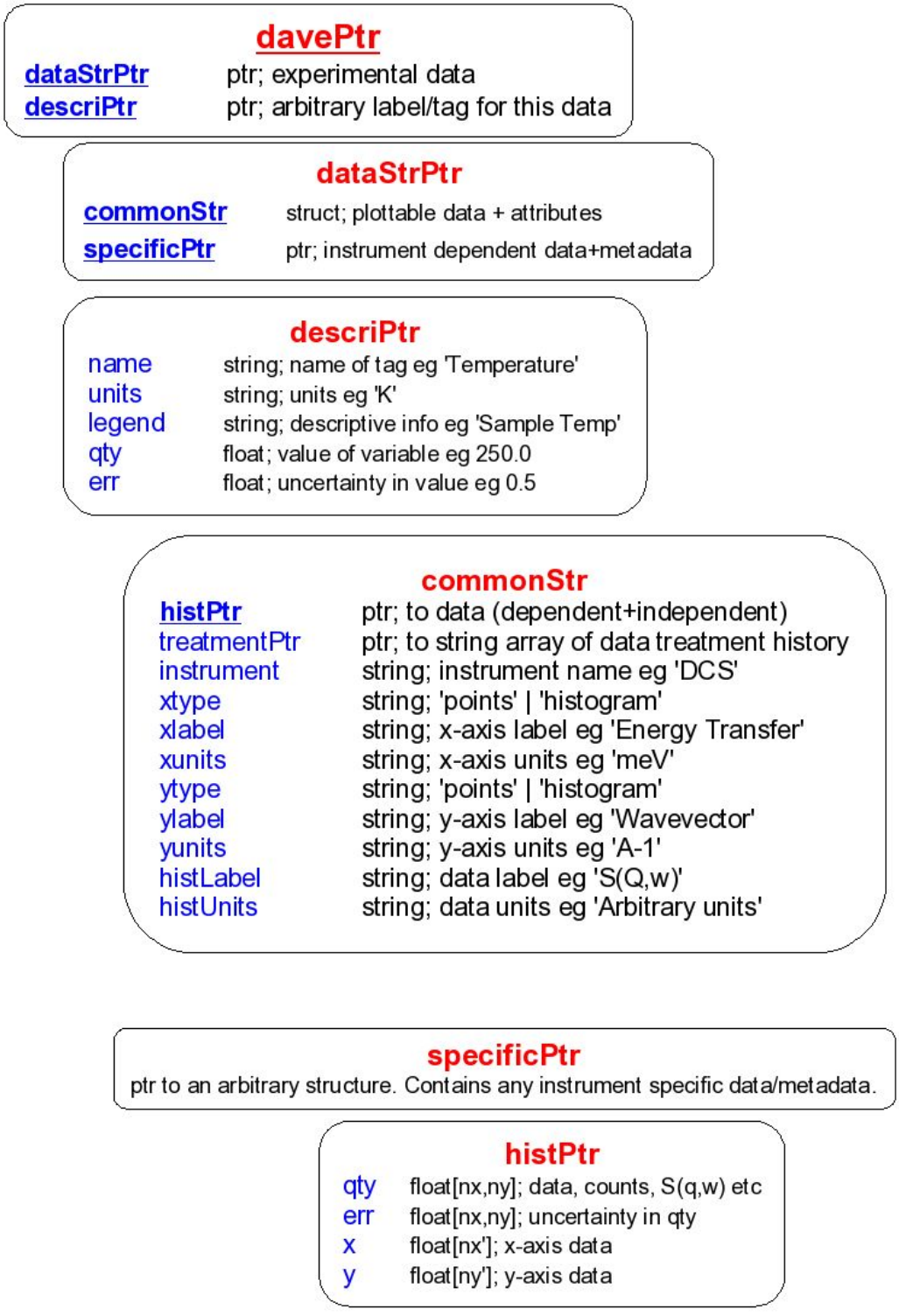
A hierarchical description of the structure now follows.

davePtr

The davePtr points to an IDL structure consisting of two other pointers: ***dataStrPtr*** and ***descriPtr***. The experimental data and all relevant metadata are stored in the dataStrPtr. The descriPtr is used for labeling or tagging the dataset.

descriPtr

The descriPtr points to a structure as described in the diagram. It is used for labeling or tagging the davePtr. This tagging can be user-defined and consists of a name, description and value. For example a user might make several measurements of a particular sample at different temperatures. The descriPtr may be used to identify these measurements as part of a series. In this case the (*descriPtr).name and (*descriPtr).units fields for all measurements could be 'Temperature' and 'K' respectively while the actual temperature for each measurement would be stored in the corresponding (*descriPtr).qty field. Although the descriPtr is optional, if present some applications within DAVE do make use of it. For example the Data Browser (visualization module) is able to combine datasets that belong to the same series to produce a composite dataset.


davePtr
dataStrPtr ptr; experimental data
descriPtr ptr; arbitrary label/tag for this data
dataStrPtr
commonStr struct; plottable data + attributes
specificPtr ptr; instrument dependent data+metadata
descriPtr
name string; name of tag eg 'Temperature'
units string; units eg 'K'
legend string; descriptive info eg 'Sample Temp'
qty float; value of variable eg 250.0
err float; uncertainty in value eg 0.5
commonStr
histPtr ptr; to data (dependent+independent)
treatmentPtr ptr; to string array of data treatment history
instrument string; instrument name eg 'DCS'
xtype string; 'points' | 'histogram'
xlabel string; x-axis label eg 'Energy Transfer'
xunits string; x-axis units eg 'meV'
ytype string; 'points' | 'histogram'
ylabel string; y-axis label eg 'Wavevector'
yunits string; y-axis units eg 'A-1'
histLabel string; data label eg 'S(Q,w)'
histUnits string; data units eg 'Arbitrary units'
specificPtr
ptr to an arbitrary structure. Contains any instrument specific data/metadata.
histPtr
qty float[nx,ny]; data, counts, S(q,w) etc
err float[nx,ny]; uncertainty in qty
x float[nx']; x-axis data
y float[ny']; y-axis data

dataStrPtr

The experimental data is contained within the dataStrPtr. The dataStrPtr points to a structure consisting of two fields: ***commonStr*** and ***specificPtr***. commonStr contains the plottable data and is the only mandatory component of the davePtr. The specificPtr is optional and is included as a placeholder for any instrument specific information.

specificPtr

Points to an instrument-dependent structure that can be used to store any relevant information about that instrument. The definition of the structure is unspecified and hence any information stored here can only be useful for instrument specific modules.

commonStr

The commonStr is an IDL structure designed to hold plotable data and additional basic attributes required to display it. A structure member, ***histPtr,*** holds the data itself with the remaining fields storing various useful attributes for displaying the data. The meaning of the attributes are easily understood as described in the diagram. The treatmentPtr points to a string array of arbitrary length that describes all the treatment details applied so far to the DAVE pointer. Hence, all program modules that modify a DAVE pointer should ensure that the treatmentPtr is updated accordingly with concise human readable textual information about the modification.

histPtr

The actual data is stored in the histPtr, a four member structure as indicated in the diagram. The dependent variable (eg counts) is stored in the *qty* field and the associated uncertainty in *err*. The first independent variable is stored in the *x* field. If *qty* is two dimensional then the second independent variable is stored in the *y* field. The dimensions of the fields should obey these rules:

histPtr field name	***dimension***	***dimension scale***
qty	1D or 2D	(nx) or (nx,ny)
err	1D or 2D	(nx) or (nx,ny)
x	1D	nx' where nx'=nx; if commonStr.xtype='points' nx'=nx+1; if commonStr.xtype='histogram'
y	1D	ny' where ny'=ny; if commonStr.ytype='points' ny'=ny+1; if commonStr.ytype='histogram'

2. Working with the davePtr

Once you become familiar with the davePtr structure, it is quite straightforward to directly manipulate it's contents. Simply remember:

- the syntax for accessing structure members (structure.field).

- a pointer variable has to be dereferenced to access its contents (all pointers within the DAVE pointer have a Ptr suffix).
- to be careful about inadvertently deleting heap memory within the dataset. For example never directly assign a local pointer to one in the data structure and then subsequently freeing the local pointer – this would result in deletion of content from the dataset.

A few simple examples will now be given to illustrate reading/writing from/to a DAVE pointer. In all examples it will be assume that the DAVE pointer is available as a locale variable called davePtr – the internal structure will already be properly defined for you. A separate document is available that deals with attaching an application module to the DAVE suite , obtaining a reference to the DAVE pointer and reading/writing to a file.

Example 1
Reading the plottable data (counts, errors, independent variable(s)) .

```
IDL>data  = (*(*(*davePtr).dataStrPtr).commonStr.histPtr).qty
IDL>error = (*(*(*davePtr).dataStrPtr).commonStr.histPtr).err
IDL>xval  = (*(*(*davePtr).dataStrPtr).commonStr.histPtr).x
IDL>yval  = (*(*(*davePtr).dataStrPtr).commonStr.histPtr).y
```

The local variables data, error, xval and yval should now contain *copies* of the data, associated uncertainties, x- and y-axis data. Note that if data is 1D then yval will be scalar (value 0.0) instead of a vector.

Example 2
Reading the treatment history. This contains an account of the data processing that the dataset has an undergone.

```
history = (*(*(*davePtr).dataStrPtr).commonStr.treatmentPtr)
```

The local variable, history, should contain the treatment history as a string array.

Example 3
Obtaining plot attributes.

```
IDL>instr_name = (*(*davePtr).dataStrPtr).commonStr.instrument
IDL>xtype = (*(*davePtr).dataStrPtr).commonStr.xtype
IDL>xlabel = (*(*davePtr).dataStrPtr).commonStr.xlabel
IDL>xunits = (*(*davePtr).dataStrPtr).commonStr.xunits
```

xtype will be 'points' or 'histo*gram', etc

Example 4
Updating data. counts and xdata are local variables containing the new data. Can either make direct assignments like

```
IDL>(*(*(*davePtr).dataStrPtr).commonStr.histPtr).qty = counts
IDL>(*(*(*davePtr).dataStrPtr).commonStr.histPtr).x = xdata
```

or (to emphasize a point) make use of an intermediate local variable for histPtr

```
IDL>local_histPtr = (*(*davePtr).dataStrPtr).commonStr.histPtr
IDL>(*local_histPtr).qty = counts
IDL>(*local_histPtr).x = xdata
```

Both methods are equivalent. However, in the second case, the local variable local_histPtr should not be freed (ie don't use: ptr_free, local_histPtr) since this is the same heap memory that is being used by the histPtr field within davePtr.

Example 5

Appending information to the treatmentPtr. This should be done whenever a noteworthy modification is made to the dataset.

```
IDL>old_rec = (*(*(*davePtr).dataStrPtr).commonStr.treatmentPtr)
IDL>new_rec = ['First additional line - counts scaled by 2',
               'Second line as required, etc',
               '----------------------------------------']
IDL>new_rec = [old_rec,new_rec]
IDL>(*(*(*davePtr).dataStrPtr).commonStr.treatmentPtr) = new_rec
```

String arrays are used for storing treatment information. The style is arbitrary but the language should be clear and concise.

While it is straightforward to work with the DAVE pointer, the syntax is very verbose and as such it is easy to make mistakes. For these reasons, a set of functions have been created for writing to and reading from a DAVE pointer. Brief references for these functions now follow.

function create_dave_pointer, davePtr, instrument=instrument, qty=qty, qtunits=qtunits, $
qtlabel=qtlabel, err=err, xvals=xvals, xtype=xtype, xunits=xunits, xlabel=xlabel, yvals=yvals, $
ytype=ytype, yunits=yunits, ylabel=ylabel, specificstr=specificstr, treatment=treatment, $
dname=dname, dunits=dunits, dlegend=dlegend, dqty=dqty, derr=derr, ermsg=errmsg

- Used for creating a davePtr and/or modifying its contents
- all keywords except ermsg specify input quantities
- davePtr parameter and qty keyword are required. If undefined on entry, then a new heap memory with a valid DAVE pointer structure will be created.

function get_dave_pointer_contents, davePtr, instrument=instrument, qty=qty, qtunits=qtunits, $
qtlabel=qtlabel, err=err, xvals=xvals, xtype=xtype, xunits=xunits, xlabel=xlabel, yvals=yvals, $
ytype=ytype, yunits=yunits, ylabel=ylabel, specificstr=specificstr, treatment=treatment, $
dname=dname, dunits=dunits, dlegend=dlegend, dqty=dqty, derr=derr, ermsg=errmsg

- Used for reading the contents of the specified davePtr
- all keywords specify output quantities
- davePtr parameter is required

Both functions return a 1 if successful or 0 otherwise. Meaning of parameter/keywords:

davePtr	DAVE pointer. For create_dave_pointer() it can either be an input or output parameter. For get_dave_pointer_contents() it is an input parameter. davePtr must be a valid DAVE pointer when it is used as an input parameter.
instrument	string variable describing instrument.
qty	double or float array containing the data variable
qtunits	string variable specifying the units of qty
qtlabel	string variable specifying the plot label for qty
err	double of float array containing the error associated with qty. It must have the same dimension and size as qty.
xvals	double or float array containing the first independent variable for qty. If not provided then a simple index array is determined based on the size of qty.
xtype	string variable specifying if the first independent variable, xvals, is "POINTS" or "HISTOGRAM".
xunits	string variable specifying the units of xvals.
xlabel	string variable specifying the plot label for the first independent variable, xvals.
yvals	y-axis equivalent of xvals
ytype	y-axis equivalent of xtype
yunits	y-axis equivalent of xunits
ylabel	y-axis equivalent of xlabel
specificstr	structure containing any instrument specific information to be include in the DAVE pointer. Can contain any variable type in its fields.
treatment	string array of any length containing the treatment of the data.
dname	string name of the davePtr tag (descriPtr).
dunits	string units of the davePtr tag (descriPtr).
dlegend	string description of the davePtr tag (descriPtr).
dqty	value of the davePtr tag (descriPtr).
derr	uncertainty in the value of the davePtr tag (descriPtr).
ermsg	output keyword. Contains error message if the function is unsuccessful.

Use of these functions can best be illustrated by revisiting the previous examples.

Example 1
Reading the plottable data and plot attributes.

```
IDL>status = get_dave_pointer_contents(davePtr $
```

```
            ,qty=data, err=errors, xvals=xval $
            ,xtype=xtype, xlabel=xlabel, xunits=xunits, ermesg=ermsg)
```

If the function fails (status=0) then ermsg should be examined for the error message.

Example 2
Reading the treatment history. This contains an account of the data processing that the dataset has an undergone.

```
IDL>status = get_dave_pointer_contents(davePtr $
         ,treatment=history, ermesg=ermsg)
```

The local variable, history, should contain the treatment history as a string array.

Example 3
Updating data. counts and xdata are local variables containing the new data.

```
IDL>status = create_dave_pointer(davePtr $
            ,qty=data, xvals=xdata, ermsg=ermsg)
```

Example 4
Appending information to the treatmentPtr. This should be done whenever a noteworthy modification is made to the dataset.

```
IDL>status = get_dave_pointer_contents(davePtr, treatment=cur_hist)
IDL>app_hist = ['First additional line - counts scaled by 2',
                'Second line as required, etc',
                '----------------------------------------']
IDL>status = create_dave_pointer(davePtr, treatment=[cur_hist,app_hist])
```