HOW GMT OBJECTS ARE DEALT WITH INTERNALLY

The various GMT containers (**families)**, i.e., grids, datasets, textsets, palettes, images, postscript, matrix, vector are created, duplicated, set as input sources (for “reading”) or output destinations (for “writing”) and finally read or written as indicated or given as input or output to modules. All such containers will be registered with the API, which maintains a GMTAPI\_DATA\_OBJECT array with one item per container. Each object has two void pointers: resource and data. In general, we read from resources and write to data but this is unclear at times. Here is the current status:

**GMT\_Create\_Data**: Allocates a new container with or without data (depending on mode). For GMT\_IS\_GRID and GMT\_IS\_IMAGE it can be called twice: First to get a container, second to allocate data (by passing the address that was returned the first time). All other families can only be called once. When the container is created, it is registered via a new GMTAPI\_DATA\_OBJECT and we set both **data** and **resource** to point to that container. If direction is GMT\_OUT then also we set **messenger** = true. This usually means the container is a dummy (empty) and it will be deleted and replaced by a container produced in a module. Alternatively, if a container is actually given with preallocated data array then GMT will not allocate space but use the given array for returning the data (this is either via a matrix or vectors). GMT cannot reallocate this array to hold more space so there are some limitations imposed. *Actions: Only set* ***data*** *if input and* ***resource*** *if output.*

**GMT\_Duplicate\_Data**: This takes an input container (we don't check if registered or not) and duplicates the container and optionally any data arrays (determined by mode). The new container is registered and attached to the objects **data** pointer, with **resource** = NULL. *Actions: Insist input is registered.*

**GMT\_Open\_VirtualFile**: For **input** it requires a valid container. If not registered by GMT\_Create\_Data it will be registered now. We then set **resource** = data and **data** = NULL. Method is REFERENCE by default but may be demoted to DUPLICATE if the container data type does not match the destination, which can happen when we pass a MATRIX to represent a GRID, for example. For output, it may optionally take a container with a user array, but most often it is NULL and GMT will allocate and return the container requested. For custom programs wanting to pass and receive basic arrays it is common to open datasets or grids using the GMT\_VIA\_MATRIX or GMT\_VIA\_VECTOR modifications to family. To have GMT allocate the right array for MATRIX/VECTOR you add the relevant GMT\_VIA\_<type> to the geometry flag, e.g., GMT\_IS\_SURFACE|GMT\_VIA\_ULONG will create an uint64\_t array on output. *Actions: Update documentation on virtual files to explain GMT\_VIA\_<type>, and fix any issues in test\*.c programs.*

**GMT\_Read\_Data**: Whatever the resource (filename or pointer to a virtual file container), the resource is read into memory and placed in the objects **data** pointer. For an input file, we set **resource** as well (why?). And if via is true (importing into GRID or DATASET from MATRIX/VECTOR) then we leave **data** = NULL (why?). Actually, a **data** = container statement is executed under the various method cases, so perhaps **data** is always set. *Actions: Check if the via test is pointless and should be removed*.

**GMT\_Write\_Data**: Given the input container we write to files or memory. File is straightforward. For the other methods, the memory ends up being attached to the objects **resource** pointer, and the **data** pointer is set to NULL.

**GMT\_Read\_VirtualFile**: This simply takes the virtual file name and returns a pointer to the container. This is how a module will access a container “written” to a virtual file. It will set **data** = **resource** then **resource** = NULL, and return **data**.

**GMT\_Close\_VirtualFile**: For input we set **data** = **resource** and **resource** = NULL again. *Actions: Add debug message*.

**GMT\_Init\_VirtualFile**: This just resets internal variables so the container can be used again. *Actions: Add debug message*

**GMT\_Destroy\_Data**: Takes address to a container and frees its content and removes (unregisteres) the object. However, destroy may fail benignly if it is told to free a container that was created at a higher module level that where it is called. We cannot delete containers created outside the level we are at except **messenger** objects.