HOW GMT OBJECTS ARE DEALT WITH INTERNALLY

Paul Wessel, Updated July 22, 2017

The various GMT containers (**families)**, i.e., grids, datasets, textsets, palettes, images, postscript, matrix, and vector are created, duplicated, set as virtual input sources (for “reading”) or virtual output destinations (for “writing”) in module calls. Such containers are registered with the API, which maintains an internal GMTAPI\_DATA\_OBJECT array with one item per container. Among such object’s metadata is a void\* pointer called **resource**. When modules read or write data this pointer is set to the allocated memory. Here is a brief overview of how the main API functions work:

**GMT\_Create\_Data**: Allocates a new container with or without data (depending on mode). For GMT\_IS\_GRID and GMT\_IS\_IMAGE it may be called twice: First to get a container with a header, second to allocate data (you must pass the address returned by the first call). For other families you must call it once. The container is created and registered via a new GMTAPI\_DATA\_OBJECT and we set its **resource** to point to that container, which is returned. If direction is GMT\_OUT then also we set **messenger** = true. This usually means the container is a dummy (empty) that will be replaced by a container created deep inside a module. This is how memory allocated in a module survives the module. Alternatively, if a container is given a pre-allocated array then GMT will not allocate space but use the given array for returning the data (via a matrix or vectors). GMT will not reallocate this array if it needs more space.

**GMT\_Duplicate\_Data**: Takes an input container (registered or not), duplicates it and optionally allocates and duplicates data arrays (determined by type and mode). The new container is registered and attached to the object’s **resource** pointer which is also returned back to the caller.

**GMT\_Open\_VirtualFile**: For **reading** it requires a valid container. If not known it will be registered and we set **resource** to point to the container. Method is REFERENCE by default but may be demoted to DUPLICATE if the container shape or type do not match destination. For **writing**, it may *optionally* take a container with a pre-allocated user array, but most often it is NULL and the module will allocate and return the container produced. User programs wanting to pass and receive basic arrays must open datasets or grids as virtual files by adding the GMT\_VIA\_MATRIX or GMT\_VIA\_VECTOR modifications to family. To make GMT allocate the right data type array for MATRIX or VECTOR you may add the relevant GMT\_VIA\_<*type*> constant to the geometry argument, e.g., GMT\_IS\_SURFACE | GMT\_VIA\_ULONG will create an **uint64\_t** array on output. The default is determined by GMT\_EXPORT\_TYPE [GMT\_DOUBLE].

**GMT\_Read\_Data**: Given the input argument (actual or virtual file name), the source is read into memory and placed in the objects **resource** pointer, which is also returned to the caller. You may call it twice for grids and images.

**GMT\_Write\_Data**: Given a container and the output destination (actual or virtual file name), we write data to a file or memory. Other than for actual file destinations, the memory ends up being attached to the objects **resource** pointer.

**GMT\_Read\_VirtualFile**: Takes a virtual file name and returns the **resource** pointer to the corresponding container. This is how we access a container “written” to a virtual file by a GMT module.

**GMT\_Close\_VirtualFile**: Not clear what its purpose is now, but it is nice to close what was opened.

**GMT\_Init\_VirtualFile**: Resets internal variables so the container may be read again. *Actions: Add debug messages.*

**GMT\_Destroy\_Data**: Takes a container address and frees its content and removes (and un-registers) the object. However, this may fail benignly if we pass a container created at a higher module level than where GMT\_Destroy\_Data is called. We cannot delete containers created above the level we are presently at except for dummy **messenger** output objects.