NAME

gft_close, gft_close_all, gft_set_d_type, gft_write, gft_write_brief, gft_write_full, gft_read_brief, gft_read_full, gft_read_shape, gft_read_id_str_p, gft_read_id_int_p, gft_read_id_float_p, gft_read_2idata, gft_read_1idata, gft_read_idata, gft_write_id_str_p, gft_write_id_int_p, gft_write_id_float_p, gft_write_2idata, gft_write_1idata, gft_write_idata

RNPL HDF routines

SYNOPSIS

```
#include <bbhutil.h>


BBH_I gft_close(const BBH_C *nm);
BBH_I gft_close_all(BBH_V);


BBH_I gft_set_d_type(BBH_I type)


BBH_I gft_write(const BBH_C *func_name, BBH_D time,
BBH_I *shape, BBH_I rank, BBH_D *data);
BBH_I gft_write_brief(const BBH_C *func_name, BBH_D time,
BBH_I *shape, BBH_I rank, BBH_D *data);
BBH_I gft_write_bbox(const BBH_C *func_name, BBH_D time,
BBH_I *shape, BBH_I rank, BBH_D *coords, BBH_D *data);
BBH_I gft_write_full(const BBH_C *func_name, BBH_D time,
BBH_I *shape, BBH_C **cnames, BBH_I rank,
BBH_D *coords, BBH_D *data );


BBH_I gft_read_brief(const BBH_C *file_name, BBH_I level, BBH_D *data);
BBH_I gft_read_full(const BBH_C *file_name, BBH_I level,
BBH_I *shape, BBH_C **cnames,
BBH_I rank, BBH_D *time, BBH_D *coords,
BBH_D *data);
BBH_I gft_read_shape(const BBH_C *file_name, BBH_I *shape);


BBH_I gft_read_id_str_p(const BBH_C *file_name, const BBH_C *param_name,
BBH_C **param, BBH_I nparam);
BBH_I gft_read_id_int_p(const BBH_C *file_name, const BBH_C *param_name,
BBH_I *param, BBH_I nparam);
BBH_I gft_read_id_float_p(const BBH_C *file_name, const BBH_C *param_name,
BBH_I *param, BBH_I nparam);
BBH_I gft_read_2idata(const BBH_C *file_name, const BBH_C *func_name,
BBH_I *shape, BBH_I rank,
BBH_D *datanm1, BBH_D *datan);
BBH_I gft_read_1idata(const BBH_C *file_name, const BBH_C *func_name,
BBH_I *shape, BBH_I rank, BBH_D *datan);
```

```
BBH_I gft_read_idata(const BBH_C *file_name, const BBH_C *func_name,
BBH_I *shape, BBH_I rank, BBH_D *data);


BBH_I gft_write_id_str_p(const BBH_C *file_name, const BBH_C *param_name,
BBH_C **param, BBH_I nparam);
BBH_I gft_write_id_int_p(const BBH_C *file_name, const BBH_C *param_name,
BBH_I *param, BBH_I nparam);
BBH_I gft_write_id_float_p(const BBH_C *file_name, const BBH_C *param_name,
BBH_D *param, BBH_I nparam);
BBH_I gft_write_2idata(const BBH_C *file_name, const BBH_C *func_name,
BBH_I *shape, BBH_I rank,
BBH_D *datanm1, BBH_D *datan);
BBH_I gft_write_1idata(const BBH_C *file_name, const BBH_C *func_name,
BBH_I *shape, BBH_I rank, BBH_D *datan);
BBH_I gft_write_idata(const BBH_C *file_name, const BBH_C *func_name,
BBH_I *shape, BBH_I rank, BBH_D *data);
```

DESCRIPTION

BBHUtil provides a set of C and FORTRAN callable functions which read and write the HDF files which are read and written by rnpl generated programs. These include initial data files as well as evolution output files.

File names may be altered from the parameter in the following ways: spaces are killed, non-alphanumeric characters are replaced by _, and .hdf is appended if it is not already there.

gft_close closes the HDF file whose name is passed as a parameter.

gft_close_all closes all open HDF files.

gft_set_d_type will set the grid function type which is written by the gft_write* calls. The default type is DFNT_FLOAT64. The only other valid type is DFNT_INT32.

gft_write, gft_write_brief will write the grid function given in data to the evolution HDF file whose name is formed from the func_name. Func_name is the name of the grid function. Time gives the time at which the grid function had the values given in data. The rank and shape provide the size of the grid function. The coordinates are named x,y,z and the coordinate values are integers running from 1 to shape.

gft_write_bbox performs the same function as gft_write. In addition, the coordinate values are taken from the bounding box coords. Coords is a one dimensional array the whose first two values provide the min and max for the first coordinate, the next two values provide the min and max for the second coordinate, etc. In general, the minimum for the nth coordinate is stored in position 2(n-1) and the maximum is in position 2(n-1) + 1.

gft_write_full works as the above except the coordinate names are given explicitly in cnames and the coordinate values are given explicitly in coords. The first shape[0] values of coords are the values of the first coordinate, the next shape[1] values are for the second coordinate, etc.

gft_read_brief returns a grid function from the file given by file_name. Level runs from 1 to the number of time levels stored in the file.

gft_read_full takes the file name, desired time level, and rank and returns complete information including the grid function, time, shape, coordinate names, and coordinate values. Storage for the returned values must be preallocated.

gft_read_shape returns the shape of the data sets in an rnpl evolution HDF file.

gft_read_id_str_p, gft_read_id_int_p, and gft_read_id_float_p are for reading parameters from initial data HDF files. The routines take the file name, parameter name, and number of parameters (array size) and return the parameter values. Sufficient storage must be preallocated and passed in.

gft_read_2idata reads two time levels of initial data from an initial data HDF file. File_name is the name of the initial data file. Func_name is the name of the grid function. Shape and rank contain the size of the data sets. Datan is the data set with the later time, and datanm1 is the data set at the earlier time. Storage must be preallocated.

gft_read_1idata works the same way except for grid functions with only one time level of initial data.

gft_read_idata will read one time level of data from an initial data file for a function with any number of time levels. Func_name must carry the time level information as well as the function name. For instance if the grid function name is "Phi" then the first time level (earliest) would be requested with a func_name of "Phi[0]" while the next would be "Phi[1]" and so on.

gft_write_id_str_p, gft_write_id_int_p, and gft_write_id_float_p work like their read counter parts, except they write data to an initial data file.

gft_write_2idata, gft_write_1idata, and gft_write_idata work like their read counter parts, except they write data to an initial data file.

All functions return 1 upon success and 0 on failure. Error messages are sent to stderr. The routines take care of opening or creating files as necessary. The evolution data write routines (gft_write[brief,bbox,full]) will check the environment variable GFT_HDFAPPEND. If this variable is set to 'on' then existing evolution files will be appended to instead of overwritten.

FILES

bbhutil.h	Header file for BBHUtil routines.
libbbhutil.a	Library file.

SEE ALSO

rnpl(1)

The RNPL Reference Manual
The RNPL User's Guide