

NAME

data_pipe() or array_pipe() – integrate external programs into spec

DESCRIPTION

spec's data_pipe() and array_pipe() functions allow integration of external code with spec. With the data-pipe facility, spec sends information to the external program, allows the external program to execute for a time, and then receives information back from the external program. The information can be in the form of a string or a number, and can also include the contents of a spec data group or data array. The handshaking and data transfer between spec and the data-pipe program is done in an overhead module included in the spec distribution that is linked with the external code.

Although data_pipe() was originally designed to work with the data group facility (see the *data* help file), both data_pipe() and array_pipe() can be used with data group or data array arguments (see the *arrays* help file) interchangeably. There is no difference in implementation. In the following discussion, the data_pipe() name will be used.

THE spec SIDE

From spec, access to the data-pipe facility is through the data_pipe() function called from the user level.

The timeout feature was implemented in release 5.07.01-1. If a timeout is set and the data-pipe process doesn't respond within the given timeout value, the process is terminated and the data_pipe() call returns -1. If the data-pipe process terminates on its own, spec display a message to that effect and the data_pipe() call returns -1.

data_pipe("?") – Lists the currently running data-pipe processes with name and process id.

data_pipe("timeout") – Returns the global timeout value. When the timeout is zero, the data-pipe process can take as long as it needs.

data_pipe(program, "timeout") – Returns the timeout value for the named data-pipe program.

data_pipe("timeout", sec) – Set the global timeout to sec seconds. A value of zero means no timeout is in effect. The global timeout is used for each new data_pipe() program.

data_pipe(program, "timeout", sec) – Set the timeout for the data-pipe process program to sec seconds, overriding the global default value.

`data_pipe(program, "kill")` – Kills the process associated with *program*.

`data_pipe(program [, args [, grp_out|arr_out [, grp_in|arr_in]])` – Initiates or resumes synchronous execution of the special process named *program*. If *program* contains a / character, then it contains the complete absolute or relative path name of the program to run. Otherwise the program must be in the `SPECD/data_pipe` directory, where `SPECD` is the built-in `spec` variable containing the path name of `spec`'s auxiliary file directory. You can use the string "." for *program* as an abbreviation for the same program name as used in the last call to `data_pipe()`.

The string value of *args* is made available to the user code in the program as described in the next section.

The optional arguments *grp_out* and *grp_in* are data group numbers. If *grp_out* is present, the contents of that group are sent to the data-pipe program. If *grp_in* is present, it is the number of the data group that will receive values from the data-pipe program. The data-pipe program configures the size of *grp_in* for an implicit call to `data_grp()` within `data_pipe()`. If the *grp_in* argument is absent, `spec` will not receive data-group data from the data-pipe program. If *grp_out* is also absent, group data won't be sent to the data-pipe program. Even without group arguments, the data-pipe program can still return values to `spec` in the form of assigning a number or string return value to `data_pipe()`.

Either or both of the data group arguments can be replaced with the array arguments *arr_out* and *arr_in*. The arrays referred to by these arguments must be the data arrays declared explicitly with the `array` keyword. When sending array data to the data-pipe program, the array data is first converted to double precision floating point format. The received data is always double, but is converted to fit the declared data type of *arr_in*. Only as much data as will fit into the array will be assigned. The number of columns in *arr_in* should match the width of the data sent over by the data-pipe program. If not, the data will still be assigned to the array, but will be misaligned.

Prior to `spec` release 4.03.13, only one `data_pipe()` function could be active at a time.

THE PROGRAM SIDE

The user C code can be compiled and linked using the command

```
dpmake program [ UOBJ=... ] [ LIBS=... ] [ opt_make_args ]
```

The command `dpmake` is a short shell script which invokes the `make` utility using the makefile `data_pipe.mak` in the `SPECD/data_pipe` directory. The file `program.c` will be compiled and linked with the data-pipe overhead module, and the result placed in an executable file named `program`. If additional object modules or libraries need to be linked, they can be specified with the `UOBJ=` or `LIBS=` parameters. If the tools provided are not sufficient, you can create custom makefiles based on the distributed `data_pipe.mak`.

After linking `program`, move it to the `SPECD/data_pipe` directory for easy access by all users.

The subroutines available from the user C code portion of the data-pipe program are described below.

The User C Code

The skeleton user C-code part of the data-pipe program should contain the following:

```
#include <user_pipe.h>

user_code(argc, argv)
char      **argv;
{
    ...
}
```

The include file `user_pipe.h` contains declarations of the subroutines available in the C code. The file resides in the `SPECD/data_pipe` directory.

The subroutine `user_code()` is called by the overhead part of the data-pipe program each time `data_pipe()` is invoked in `spec`. The parameter `argc` is set to the number of space-delimited words present in the string value of the `args` parameter to `data_pipe()`. The parameter `argv` is an array of character pointers that point to each of the `argc` space-delimited words in the `args` string. Alternatively, the `get_input_string()` function (see below) returns the `args` string in its entirety.

The `user_code()` routine will be called every time the `data_pipe()` function is called from `spec`. The data-pipe program does not exit between calls of `user_code()`, so you should be careful about allocating memory or opening files each time `user_code()` is called without freeing the memory or closing the files each time `user_code()` returns. Alternatively, you can

make sure such things are only done the first time `user_code()` is called.

Accessing Command Line Arguments

Besides the `argc, argv` technique for accessing the `args` typed in the `data_pipe()` call, the following function is available (as of `spec` release 4.03.13):

```
char *get_input_string() – Returns a pointer to memory holding a copy of the second argument args entered with the call to data_pipe().
```

Reading the Group/Array Data In the C Code

If `data_pipe()` is sending a data group or array to the user code, the following subroutines provide access to the data parameters and values.

```
int get_group_number() – Returns the group number specified as the data_pipe() grp_out argument. A -2 is returned if an array was specified. A -1 is returned if neither data group or array was specified.
```

```
int get_group_npts() – Returns the number of points in the data_pipe() grp_out or the number of rows in arr_out.
```

```
int get_group_width() – Returns the number of elements per point in the data_pipe() grp_out or the number of columns in arr_out.
```

```
int get_input_data(double *x, int pts, int wid) – Transfers data from the grp_out or arr_out specified in the call to data_pipe() to the memory area indicated by the pointer x. The pointer x is treated as an array dimensioned as x[pts][wid]. If the data group/array has more points/rows than pts or more elements/columns than wid, only as many points/rows or elements/columns as are available in the data group/array are copied. Data from only a single element/column may be retrieved using one or more calls of get_input_elem() below. If the data in the data group from spec is float rather than double (which depends on spec's installation configuration), float-to-double conversion is done within the call to get_input_data(). The return value is the number of points/rows copied.
```

```
int get_input_elem(double *x, int pts, int el) – Transfers one element of the data from the grp_out or arr_out specified in the call to data_pipe() to the memory area indicated by the pointer x. No more than pts points are copied from element/column el of the the data group/array. If the data in the data group from spec is float rather than double (which depends on spec's installation configuration), float-to-double conversion is done within the call to
```

`get_input_data()`. The return value is the number of points/rows copied.

Sending Group/Array Data Back To spec

The following subroutines allow you to send group/array data back to `spec` when `data_pipe()` is invoked with a `grp_in` or `arr_in` argument. For a data group, the call to `data_pipe()` will implicitly call `data_grp()` to configure the size of the return group according to the parameters set in the following subroutines. For an array, the array must already be declared and dimensioned.

There are two ways to send group/array data back to `spec`. The subroutine `set_return_data()` allows you to send the entire data group in one call that passes both a pointer to the data and the data group size to the data-pipe program overhead code. Alternatively, you can use the `set_return_group()` subroutine to configure the data group/array size, followed by one or more calls to `set_return_elem()` to set one element/column of the data group/array at a time.

`int get_return_group_number()` – Returns the group number specified as the `data_pipe()` `grp_in` argument. A `-2` is returned if an array was specified. A `-1` is returned if neither data group or array was specified.

`void set_return_data(double *x, int pts, int wid, int last)` – Configures the return data group and copies the data at the same time. The pointer `x` is considered as an array of dimension `x[pts][wid]` for the purpose of transferring data to the data group. The argument `last` sets the index number of the last element added to the group, which is used by the various data analysis and plotting functions available in `spec`.

`void set_return_group(int pts, int wid)` – Configures the size of the return data group without copying data. This subroutine must be called once before calling `set_return_elem()` below.

`int set_return_elem(double *x, int pts, int el, int last)` – Copies one element to the return data group, which must have been previously configured by a call of `set_return_group()`, above. If the parameters `pts` or `el` exceed the values configured, or if the return group hasn't been configured, the subroutine returns `-1`. Otherwise zero is returned.

Setting data_pipe() Return Values From C Code

You can set the value that the `data_pipe()` function returns in `spec` from the user C code in your *data-pipe* process. You can have `data_pipe()` return a number or a string or, if necessary, reset to command level. If no explicit return value is assigned in the user C code, `data_pipe()` returns zero.

`int set_return_string(char *s)` – Sets the return value of `data_pipe()` to the string `s`. This subroutine returns `-1` if memory could not be obtained for the string `s`, otherwise it returns zero.

`void set_return_value(double v)` – Sets the return value of `data_pipe()` to the value `v`.

`void do_error_return()` – Calling this subroutine from the user C code causes control to pass back to `spec` without returning data group or array values, if they have been set. The return value of `data_pipe()` will be the value set by `set_return_value()` above, if such a value has been set, otherwise the return value of `data_pipe()` will be `-1`. This subroutine does not return.

`void do_abort_return()` – Calling this subroutine from the user C code causes control to pass back to `spec` without returning data group or array values, if they has been set. In `spec`, there is no return from `data_pipe()`, rather `spec` resets to command level. This subroutine does not return.

`void do_quit_return()` – Calling this subroutine from the user C code causes control to pass back to `spec` normally as if `user_code()` returned normally, but the data-pipe program will then exit. This subroutine does not return.

SEE ALSO

data