LabView interface for TACO - V1.5

A.Götz 14 June 2000

This document describes the LabView interface to TACO control systems. It describes the DSAPI client interface and how to write TACO device servers in the LabView graphical programming language G.

Contents

1 Getting started

In order to get started you need access to the following tools :

1. LabView on Unix (Linux, HP-UX or Solaris)
2. the LabView-TACO shared libraries lv_dsapi.so and lv_dsclass.so (only needed if you plan to write a device server in TACO) in your shared library path ($LD_LIBRARY_PATH for Linux/Solaris and $SHLIB_PATH for HP-UX)
3. start LabView (normally by typing labview) and open one of the example VI's or wire your own following the instructions in this document.

2 LabView-TACO interface

2.1 Clients

The following shared library calls have been implemented to interface TACO clients to LabView :

1. **lv_dev_putget()** - will execute a command on a device, one input and output argument is l passed, an error code is returned and a status. The call has the following calling syntax :

lv_dev_putget (char *name, char *cmd, void *argin, void *argout, long *error)

name : device name e.g. "id11/oregon/1" (passed as C string type)

cmd : command to execute e.g. "DevMoveRelative" (passed as C string type)

argin : input argument e.g. array of floats (passed as "Adapt to Type" type, this means the G program has to wire the correct input type expected by the command to the input argument e.g. if a double array is expected the LabView program has to provide a double array as input, failure to do so can result in a core dump of the LabView program ! NOTE: the input has to be wired even it is not used to a dummy type of the correct type; this is because LabView does not differentiate between not used input types when calling library functions and will not allow the program to run if an input is not wired - the RUN arrow is broken)

argout: output argument e.g. array of floats (passed as "Adapt to Type" type, see text above for advice on how to use this type. NOTE: the input has to be wired even it is an output to a dummy type of the correct type; this is because LabView does not differentiate between output only types when calling library functions and will not allow the program to run if an input is not wired - the RUN arrow is broken)

error : output error if any (passed as pointer to 32 bit integer. See NOTE above for wiring the output error)

returns : a 32 bit integer indicating the status of the command (0 = OK, -1 = NOT OK)

NOTE : refer to section on types to know which are supported for argin and argout

2. **lv_dc_devget()** - will retrieve the result of a command on a device from the data collector (the TACO data cache), one output argument is passed, an error code is returned and a status. The call has the following calling syntax :

   lv_dc_devget (char *name, char *cmd, void *argout, long *error)

   name : device name e.g. "sr/d-ct/1" (passed as C string type)

   cmd : command to execute e.g. "DevReadSigValues" (passed as C string type)

   argout: output argument e.g. array of floats (passed as "Adapt to Type" type, see text above for advice on how to use this type. NOTE: the input has to be wired even it is an output to a dummy type of the correct type; this is because LabView does not differentiate between output only types when calling library functions and will not allow the program to run if an input is not wired - the RUN arrow is broken)

   error : output error if any (passed as pointer to 32 bit integer. See NOTE above for wiring the output error)

   returns : a 32 bit integer indicating the status of the command (0 = OK, -1 = NOT OK)

   NOTE : refer to section on types to know which are supported for argin and argout

3. **lv_dev_protocol()** - change the RPC protocol on a device to TCP or UDP. NOTE: the default is UDP. Changing to TCP will make client connections more reliable and allow correct error detection (e.g. server down error). The call has the following syntax :

   lv_dev_protocol(char *name, char*protocol, long *error);

   name : name of device e.g. "id11/oregon/1" (passed as C string type)

   protocol : RPC protocol e.g. "tcp" or "udp" (passed as C string type)

   error : error if any (passed a pointer to 32 bit integer)

4. **lv_dev_timeout()** - change RPC timeout on a device. The call has following syntax :

   lv_dev_timeout(char *name, long timeout, long *error);

name : name of device e.g. "id11/oregon/1" (passed as C string type)

timeout : new timeout in milliseconds (pass as 32 bit integer)

error : error code if any (passed as pointer to 32 bit integer)

5. **lv_dev_free()** - close down all connections to a device. This call is useful for releasing access to devices not used anymore. It closes all open network connections to the device thereby also saving open file descriptors. The first call to the device will reimport the device. The call has following syntax :

   lv_dev_free(char *name, long *error);

   name : name of device e.g. "id11/oregon/1" (passed as C string type)

   error : error code if any (passed as pointer to 32 bit integer)

6. **lv_dev_error_str()** - return TACO error string corresponding to error_no. The call has following syntax :

   char *lv_dev_error_str(long error_no);

   error_no : error number

7. **lv_dev_cmd_query()** - return the list of commands supported by a device and their types. This call is mainly useful as info for the LabView programmer to know what commands and what data types are supported for a device . The call has following syntax :

   long lv_dev_cmd_query(char *device, void* cmd_list, long *error);

   device : the name of the device

   cmd_list : list of commands and their input/output types returned as an array of strings (passed as Adapt to Type)

   error : pointer to error code in case routine fails

8. **lv_dc_cmd_query()** - return the list of commands polled by the data collector for a device and their types. This call is mainly useful as info for the LabView programmer to know what commands and what data types are supported for a device in the data collector. The call has following syntax :

   long lv_dc_cmd_query(char *device, void* cmd_list, long *error);

   device : the name of the device

   cmd_list : list of commands and their input/output types returned as an array of strings (passed as Adapt to Type)

   error : pointer to error code in case routine fails

9. **lv_db_getdevexp()** - return the list of exported devices whose names satisfy the filter. The call has following syntax :

   long lv_db_getdevexp(char *filter, void* device_list, long *error);

   device : name filter of format D/M/F where either of D, F or M can be the wildcard *

   device_list : list of device names returned as an array of strings (passed as Adapt to Type)

   error : pointer to error code in case routine fails

2.2 Servers

It is possible to write TACO device servers in LabView. They work by implementing a device server loop in a part of the G program which polls the network periodically (using the lv_ds_cmd_get() call) to see if there are any client requests. Once a request is detected the LabView program has to execute it and then

return the answer to the client (using the lv_ds_cmd_put()) call. Clients and servers communicate using the TACO RPC protocol on the network. The TACO devices have to be defined beforehand in the TACO database. The following shared library calls have been implemented to write TACO device servers in G :

1. **lv_ds_init()** - create and initialise a LabView TACO device server, to be called once in a LabView device server before calling lv_ds_cmd_get(). The call has the following calling syntax :

   lv_ds_init(char *server, char*name);

   server : device server executable name e.g. "StressRigds" (passed as C string type)

   name : device server instance/personal name e.g. "id11" (passed as C string type)

2. **lv_ds_cmd_get()** - poll network to see if a client request has arrived, if yes command returned is non-zero. Can only be called after lv_ds_init() has been called. The call has the following syntax :

   lv_ds_cmd_get(long *command, void *argin);

   command : command received, can be one of DevLVIOStringDevState (1) takes array of strings as input and returns array of strings as output, DevLVIODouble (2) takes array of doubles as input and returns array of doubles as output, DevReadValue (3) returns an array of doubles as output, DevSetValue (4) takes array of doubles as input, DevState (5) and DevStatus (6) or 0 if no client request. DevState and DevStatus are handled automatically by the LabView device server

   argin : passed as "Adapt to Type".

3. **lv_ds_cmd_put()** - return result to client after completing executing of command. Must only be called after a lv_ds_cmd_get() has returned a non-zero command value. The call has the following calling syntax :

   lv_ds_cmd_put(long command, void *argout);

   command : command returned by lv_ds_cmd_get() (passed as 32 bit integer)

   argout : double array to be passed back to client if command was DevIOLVString, DevIOLVDouble or DevReadValue (passed as "Adapt to Type")

2.3 Debugging

An additional command exists to set the debugging flag in the library in order to display debugging information in a graphical window. The command is :

1. **lv_ds_debug(long debug)** : sets the debug flag. Setting the debug flag to a non-zero value switches on the debugging, zero switches it off.

3 Types

All TACO kernel types and motor types are supported. The following types are supported as input and output :

1. D_VOID_TYPE
2. D_SHORT_TYPE
3. D_LONG_TYPE
4. D_FLOAT_TYPE

5. D_DOUBLE_TYPE
6. D_STRING_TYPE
7. D_VAR_STRINGARR
8. D_VAR_SHORTARR
9. D_VAR_LONGARR
10. D_VAR_ULONGARR
11. D_VAR_FLOATARR
12. D_VAR_DOUBLEARR

The following types are supported as input only :

1. D_MOTOR_FLOAT
2. D_MULMOVE_TYPE

The following arguments are supported in output only :

1. D_LONG_READPOINT
2. D_FLOAT_READPOINT
3. D_DOUBLE_READPOINT
4. D_STATE_FLOAT_READPOINT
5. D_VAR_LRPARR
6. D_VAR_FRPARR
7. D_VAR_SFRPARR

4 Examples

There are examples of calling all the functions as well as examples of a TACO LabView device server (device_server.vi) and client (device_client.vi) are available. Study them to find out how to write your own clients and servers in LabView.

5 Known Problems

Some of the known problems with the TACO LabView interface are :

1. out of memory - sometimes LabView gives this error message when calling the lv_dev_putget() function. I don't know what this is due to. It seems to be occur when a string is passed as output. The only solution I have found is to rewire the corresponding (dummy) input for the output to another string type or to recreate it as a constant. This problem has been mostly occurring on HP-UX. If anyone finds a better explaination/solution for this problem let me know.
2. limited number of devices - the present interface is limited to 1000 TACO devices in a LabView session

3. not unregistered LabView device servers - there is no routine yet for unregistering LabView device servers. This is not a major problem. It simply means the number of reserved program numbers will go up every time a LabView device server is started.

4. blocked lv_dev_putget() call - this can happen on HPUX when trying to access a device running on a host which is not reachable from the host on which LabView is running. The present implementation will block. This problem does not occur on Linux.

5. client cannot run in same Labview session as server - the present version does not support running clients in the same Labview session as the device server.

6. too many files open - this problem can be encountered in Labview applications which have too many devices open. The latest (V1.5) version of the TACO interface sets the limit of the number of open files to the maximum operating system allowed value at the first call. If you still have problems then your application is too big ! Previous versions (<V1.5) are limited by the default limit for no. of open files e.g. 60 on HP-UX.

7. Labview hangs - this can happen when accessing a device with UDP protocol and the server/host are down. The solution is to switch to TCP protocol.

6 Future developments

This is the fifth release of the LabView TACO interface. A number of improvements been made e.g. adding support for the data base, dynamic device management, increasing limit on open files. In the future we plan to offer a VI library for TACO which will reduce the programming effort on clients even more. If any readers have ideas for other improvements they should send their comments to the author (goetz@esrf.fr) or even better add them to the source code themselves and then send the code to the author !