ANGLO-AUSTRALIAN OBSERVATORY AAO/DTCL_12
**DRAMA Software Report 12**
**Version 0.6.3**

Jeremy Bailey,Tony Farrell
15-Nov-97

# DTCL - Tcl Interface to Drama

## Contents

Revisions:

**V0.4.2 11-Oct-1994** The routine DtclMainLoop has been supercedded by DtclAppMainLoop and DtclTkMainLoop has been supercedded by DtclAppTkMainLoop. The new routines have additional arguments. The old interfaces still exist be are no longer documented.

**V5.0 08-Feb-1995** Introduced revisions section. Add new Tcl commands `DtclFindFile`, `DtclParseFile` and `after`. Add "Usage Notes" section. Document DtclBackgroundError procedure.

**V5.1 27-Mar-1995** Document `DtclCmdError`. Mention `RequestNotify`.

**V5.2 08-Dec-1995** Add `DtclErsRep` routine, `DitsSignal` and `DitsKill` Tcl commands. Add "-killarg" flag to `obey` command.

**V5.x 22-Jul-1996** Add new commands `DitsTaskIsLocal` `DtclDitsScanTasks`, `DitsSetDetails`, `DitsGetSeq`, `DitsGetContext`, `DitsPutRequest`, `DitsMsgOut`, `DitsPutAction`, `DitsPutObeyHandler`, `DitsPutKickHandler`, `DitsGetActIndex`, `DitsGetEntTransId`, `DitsGetEntName`, `DitsMessage`, `DitsGetPath`, `DitsLoad`, `DitsRequestNotify`. Add the C routines `DtclPutAction`, `DtclPutTclObeyHandler`, `DtclPutTclKickHandler`, `DtclActEndCheck` In part, this adds support for adding actions implemented in Tcl and adding them from Tcl.

**V6.1 19-Feb-1997** Add `Dtcl_Init` routine when Tcl version is greater then 7.5. Document use of this as a sharable library.

**V6.2 22-Apr-1997** Add parameter system commands `DitsGetParId`, `SdpCreate`, `SdpSet`, `SdpGet` and `SdpGetSds`. Document there use. Add `-variable` and `-varcvtcmd` arguments to `monitor` command. Add `-printcmd` option to `SdsList` command. Add `See Also:` specifications to routine and command descriptions.

**V6.3 15-Nov-1997** Add new SDS commands `SdsExtract`, `SdsDelete`, `SdsFillArray`, `SdsGet`, `SdsGetExtra`, `SdsInsertCell`, `SdsNew`, `SdsPut`, `SdsPutExtra` and `SdsResize`. Available from DTCL version 1.0.4 only. Updated documentation of `DramaLoad` command.

**V6.4 29-Oct-1998** Add details of Namespace support

# 1 Introduction

Tcl is the *tool command language* developed by John Ousterhout at Berkeley. Tcl aims to provide a standard command language which can be embedded in applications, and can be extended to add additional commands for the specific application. **DTCL** is an interface between Tcl and **DRAMA** which allows Tcl to be used as a command language for **DRAMA** and to be incorporated into **DRAMA** tasks.

Associated with Tcl is the Tk package which provides the capability to construct a graphical user interface using Tcl. Tk provides a set of widgets similar to those provided by systems such as Motif which can be created with relatively simple Tcl commands.

The Tcl language itself is documented in a book by Ousterhout, and a set of man pages, and will not therefore be described in detail here. This document describes only the additional commands provided in **DTCL**.

**DTCL** provides the following capabilities:

- The **DTCL** task can be used as a standalone command line user interface to send commands to other **DRAMA** tasks in the same way that ICL is used with ADAM. A difference is that **DTCL** is itself a **DRAMA** task and has an action which will invoke a Tcl command within it. Thus commands can be received from other tasks as well as directly from the user.

- By using the **DTCL** task with Tcl scripts it is effectively possible to code a **DRAMA** control task entirely in Tcl rather than in C. **DTCL** supports a fully event driven style of programming in which Tcl procedures are invoked as callbacks on completion of actions and other events. It also supports a simpler sequential style in which Tcl waits for completion of each action before continuing.

- The **DTCL** package allows a command line user interface to be incorporated into any **DRAMA** task with minimal changes to the code. This can be a useful way of testing a task. It also allows Tcl to be used to increase the flexibility of a task by using Tcl scripts to further configure the operation of a task written in C.

- In the same way a **DRAMA** task can easily be provided with a graphical user interface using the Tk package. The changes to the code are minimal and the graphical user interface is described by a Tcl script. User interface events such as button presses and menu selections can invoke actions in the task itself or in other tasks.

- DTCL can be used as a way of setting up a network of Tcl applications which communicate with each other through the **DRAMA** message system. Although similar capabilities are provided in Tcl with the *send* command, this relies on X windows for communication and so is only available in Tk. The *send* command also blocks while executing a remote command thus freezing the user interface of a Tk application it is running in. DTCL provides such facilities for all applications (whether or not they use Tk) and can be used in an event driven way with a completion handler being invoked when a remote command completes.

- DTCL allows the prototyping of DRAMA applications using TCL. You can create DRAMA actions which are implemented partly or completely in TCL.

There are also some limitations of the current implementation of **DTCL**

- **DTCL** currently works on UNIX, VMS and VxWorks. Note that the standard release of Tcl is only available for UNIX systems. Tcl has been ported to other systems including VAX/VMS and VxWorks, but it is not portable in the sense that the same source code will run on all systems, and ports to other systems are generally maintained by other groups and will lag behind the UNIX release when a new version is produced. In addition, the Tk extension is not supported under VxWorks.

## 2 Tcl Namespaces

Version 8.0 of Tcl introduced the concept of namespaces. This concept allows the use of simple names whilst avoiding polution of the global namespace. See the Tcl man page on namespaces for details.

From V1.2, DRAMA takes advantage of this scheme, and will by default place all DRAMA commands and variables within the "drama" namespace. A a result, the `obey` command becomes `drama::obey`. This document assumes all DRAMA Tcl commands have been imported into the global namespace.

In order to support compatiblity with TCL scripts written for older versions of DRAMA and TCL, it is possible to change the use of namespaces in several ways. First, there are two levels of defaults, the system build default and the run time default. The system build time default is to use namespaces, but can be overridden by defining the Imake variable `DtclUseGlobalNamespace` in your `drama_local.cf` file to be YES. The second level of defaults is via an environment variable (or logical name). If the environment variable `DTCL_NAMESPACES` is defined, it overrides the build time default. If the value is 0, then all DRAMA commands are placed in the global namespace. If defined as any other integer value, DRAMA commands will be placed in the "drama" namespace only.

Finially, if it possible to override the default at run time. For the dtk and dtcl programs, this is done using the `-globalnames` or `-noglobalnames` options. When using the `Dtcl_Init()` interface or loading the DRAMA shareable into a version of wish, the Tcl variable `DTCL_NAMESPACES` is used. If set to 0, DRAMA commands are placed in the global name space. If set to a positive value, the namespace is used. If set to a negative value, the system default is used.

Note that under Tcl 8.0 and later, the "drama" namespace is always defined and all the DRAMA commands exist in that namespace. The actual effect of the above options to use the global namespace is that the commands from the DRAMA namespace are imported into the global namespace. Similary, DRAMA variables are linked to variables of the same name in the global namespace using the `upvar` command.

For Tcl versions before 8.0, all DRAMA commands are accessible in the global name space only (since namespaces don't exist).

It is recommended that you use namespaces for all new work.

# 3 The DTCL task

The **DTCL** task can be run by typing the command *dtcl* at the shell prompt. There are a number of command line options that can be used to modify the behaviour. These are described in appendix A.

The task will respond with a `Dtcl>` prompt. Any Tcl command can then be entered.

## 3.1 DTCL Commands to build argument structures

**DTCL** adds a number of new commands to Tcl to provide the interface to **DRAMA**. These commands allow SDS argument structures to be built and messages of various types to be sent to tasks.

The following commands provide a Tcl analogue of the Arg routines which are callable from C.

`ArgNew` creates a new argument structure and returns its SDS identifier. The `ArgPutx` routines are essentially the same as the corresponding C routines. The following Tcl code will create a simple argument structure with a string, integer, and floating point argument.

```
Dtcl> set a [ArgNew]
Dtcl> ArgPutString $a grating 600R
Dtcl> ArgPuti  $a order 2
Dtcl> ArgPutf  $a wavelength 4500.0
```

The contents of this structure could then be listed using the command SdsList which is analogous to the C routine of the same name.

```
Dtcl> SdsList $a
ArgStructure         Struct
  grating              Char   [5] "600R"
  order                Int    2
  wavelength           Float  4500
Dtcl>
```

For simple case there is a single command which will build an argument structure from a number of strings, and return the SDS id. In this case you don't have control of the name or type of the SDS items — everything will be a string and the names will be Argument1, Argument2 etc. The command is called `args` and an example of its use follows:

```
Dtcl> SdsList [args 600R 2 4500]
ArgStructure         Struct
  Argument1            Char   [5] "600R"
  Argument2            Char   [2] "2"
  Argument3            Char   [5] "4500"
Dtcl>
```

### 3.2 DTCL Commands to Send Messages to Tasks

The following commands can be used to send messages to DRAMA tasks.

obey — Send an obey message.

kick — Send a kick message.

pset — Send a parameter set message.

pget — Send a parameter get message.

control — Send a control message.

monitor — Send a monitor message.

The obey command can be used as follows to invoke the CONFIGURE action in the SPECTROGRAPH task. The argument structure built previously is passed via the Tcl variable `a`.

```
Dtcl> obey SPECTROGRAPH CONFIGURE $a
Dtcl>
DTCL:Action "CONFIGURE", Task "SPECTROGRAPH", completed.
```

Following the **DRAMA** philosophy the obey command does not block and returns immediately thus putting out a new `Dtcl>` prompt for the next command. The action finishes some time later and the automatic handling of this, puts out the Action completed message.

When carrying out simple tests this behaviour is often inconvenient and it is better to wait until the action is completed before accepting subsequent commands. This can be achieved by adding the `-wait` option to the command.

```
Dtcl> obey SPECTROGRAPH CONFIGURE $a -wait
DTCL:Action "CONFIGURE", Task "SPECTROGRAPH", completed.
Dtcl>
```

Now the action completes before the `Dtcl>` prompt is output for the next command. Please see section 7.1 for more comments on this option.

An alternative to the `-wait` option is to specify a Tcl callback procedure to be called when the action completes. This is achieved with he `-complete` option. The argument to the `-complete` option is a Tcl command to be invoked on completion of the action. Typically his will be the name of a completion handling procedure.

```
Dtcl> proc CompletionHandler {} {
Dtcl>   puts stdout "Completion Handler Called"
Dtcl> }
Dtcl> obey SPECTROGRAPH CONFIGURE $a -complete CompletionHandler
Dtcl>
DTCL:Action "CONFIGURE", Task "SPECTROGRAPH", completed.
Completion Handler Called
```

The `-complete` option is only one of several event handling options which can be added to `obey` and similar commands. The others are as follows:

-success — called when an action completes succesfully. A success handling procedure should have a single argument in which it will receive the SDS identifier of the argument structure returned in the completion message.

-error — called when an action completes with an error. An error handling procedure should have a single argument in which it will receive the status code associated with the error.

-trigger — called when a trigger message is received from the action. A trigger handling procedure should have a single argument in which it will receive the SDS identifier of the argument structure returned in the trigger message.

-info — called when an information or error message is received from the action. An info handling procedure should have a single argument in which it will receive the SDS identifier of the argument structure returned in the message. This will be a structure of MsgStructure or ErsStructure type (see DITS specification).

-variable — Not a handler but related. This option applies to the `monitor` command only. It specifies an array variable. The value of the parameter being monitored will be inserted in the array element corresponding to the name of the parameter.

-varcvtcmd — Only used in conjuction with `-variable`. Allows you to convert the value before setting the variable. For example, you might have a value which is in radians but you wish to display it in a tk label widget as hours:minutes:seconds. You can use the combination of these options to do so automatically - converting the value to the right units and then putting it in a variable which is displayed in the label widget.

All these handlers will replace the default handler which causes a message to be put out on standard output. Thus dummy handlers could be used to suppress the "Action Completed" and similar messages. Various other options are available to these commands - see Appendix C.

### 3.3 Dtcl Variables

Five Global TCL variables are defined and used by **DTCL**. These are:

MessageBytes — The number of bytes to allocate for sending messages to other tasks.

MaxMessages — The max number of messages of size `MessageBytes` which may be sent at one time.

ReplyBytes — The number of bytes to allocate for reply messages.

MaxReplies — The max number of replies of size `MaxReplies` to allocate space for.

Logging — Enable/Disable logging of detailed information about messages sent and received from other tasks. Only affects new transactions, not outstanding transactions. Default is 0 which disables Logging. Set to 1 to enable Logging.

The first four of these variables correspond to the arguments of similar names to the Dits routine **DitsGetPath()**. See Dits documentation for more details.

These variables have default values set by the particular program calling **DTCL**, and may also be set by command line options in most programs.

When you change the value of these variables, you change the buffer sizes used in future message commands (obey etc.)[1].

Several additional variables, normally provided in other Tcl applications are also setup. For example, argc, argv0 and args are provided.

### 3.4 Loading Tasks

TCL provides the ability to run unix program on the local machine using the `exec` command. Drama also provides program running abilities but with some important differences. The following features are provided by Drama:

- It allows programs to be loaded and run on remote machines using Drama networking facilities.
- If the program loaded with the Drama facilities is a Drama program, then you can be notified when it registers itself with Drama, ensuring that you only attempt to communicate with it when it is ready.
- You will be notified when the program exits or if it fails to load.

The `DramaLoad` command provides these facilities. For example

```
Dtcl> DramaLoad DITS_LIB:ticker
DTCL:Program "DITS_LIB:ticker" loaded and registered on node aaossc as TICKER.
Dtcl> obey TICKER EXIT
DTCL:Action "EXIT", Task "TICKER", completed.
DTCL:Program "DITS_LIB:ticker" on node aaossc exited with status ok.
```

The first argument to the command is the name of the program to run. As with the message sending commands, various event handling options can be specified. Additionally, various DramaLoad options are available. See the command description for full details.

Note, in older versions of Drama we used the `load` command, but as this was later used by Tcl to load shareable libraries into Tcl, we changed to using `DramaLoad`. If you use the Tcl `load` command when you actually wanted the `DramaLoad` you will get a result like this

```
Dtcl> load DITS_LIB:ticker
couldn't load file "DITS_LIB:ticker":
    ld.so.1: ./dtcl: fatal: DITS_LIB:ticker: can't open file: errno=2
```

[1] In the current version of Drama, the message sizes are only used in the first communication with a task of a given name.

### 3.5 DTCL Actions and Parameters

The **DTCL** task has two parameters and one action. The parameters are as follows:

DTCL_PROMPT — A character string specifying the prompt string to be used by **DTCL**. The prompt can be changed by changing the value of this parameter.

DTCL_ENABLE — An integer value which is 1 if the user interface is currently enabled and 0 if it is disabled. This value should not be changed.

The one action is DTCL_COMMAND. This action causes its character string argument to be executed as a Tcl command in the Tcl interpreter of the task. Any output or error messages will be returned to the task invoking the action rather than to the **DTCL** user interface.

Since a task can send messages to itself, it can set and get its own parameters. Thus a **DTCL** task can change its prompt string by issuing the folowing command:

```
Dtcl> pset DTCL DTCL_PROMPT [args ">> "] -wait
DTCL:Set of Paramater "DTCL_PROMPT", Task "DTCL", completed.
>>
```

### 3.6 Adding new actions

You can add new actions to **DTCL** using the normal DRAMA C level calls, such as `DitsPutActions()`. In addition, you can add new actions which are implemented in Tcl. I.e, A Tcl command is invoked when when the DRAMA action is invoked. This technique allows, amongst other things, quick prototyping of DRAMA tasks. You can mix in one task actions implemented in C and actions implemented in Tcl. You can even mix the use of Tcl and C in the implementation of a given action. This allows the most compute intensive operations to be converted to C whilst leaving the rest of a program implemented in Tcl.

To add a new DRAMA action from a Tcl script, use the command `DitsPutAction`. This allows you to specify the name of the DRAMA action and the Tcl commands to be invoked to handle Obey and Kick messages specifying this action name. For example, the following creates a new action named `LIST` which will list its action argument using `SdsList`.

```
DitsPutAction LIST -obey {SdsList DitsArgument}
```

You can also add and access parameters. `SdpCreate` is used to create new paramters whilst `SdpSet`, `SdpGet` and `SdpGetSds` are used to access the task's own parameters.

```
Dtcl> SdpCreate PARAM1 INT 1
Dtcl> SdpCreate PARAM2 STRING "hi there"
Dtcl> SdpCreate PARAM3 FLOAT 2.0
Dtcl> SdpCreate PARAM4 UINT 3
Dtcl> SdpPut PARAM1 10
Dtcl> SdpGet PARAM1
10
```

As a slightly more complex example, consider the DRAMA program `ticker`. This program is a simple timing and test program which is included as part of the DRAMA software. Normally it is used with the `tocker` program to test the the basic DRAMA software is working an get an idea of its performance. `tocker` obeys the action `TICK` to the task `TICKER` a number of times, specified as `tocker`'s argument. Both tasks report the time taken. `tocker` then sends the `EXIT` action to `ticker` and exits itself. For example,

```
ticker &
tocker 10000
TOCKER:connected to task TICKER
TOCKER:Total time for 10000 ticks is   9.88s, Ave Time = 0.9876ms
TICKER:Total time for 10000 ticks is   9.89s, Ave Time = 0.9886ms
```

It is easy to reimplement `ticker` in Tcl. For example

```
DitsPutAction TICK -obey TickerProc
DitsPutAction EXIT -obey exit
proc TickerProc { } {
}
```

This is run like this-

```
dtcl -t ticker.tcl -n TICKER -nostdin &
tocker 10000
TOCKER:connected to task TICKER
TOCKER:Total time for 10000 ticks is  17.40s, Ave Time = 1.7399ms
```

The `-n` option to `dtcl` changes the name the task registers as. The `-nostdin` option tells `dtcl` not to read from standard input. You can see here that the `dtcl` version of `ticker` runs at half the speed that the C version does. This is really not too bad considering the large amount of extra work involved (convertions to strings, look up of command procedure etc.)

All the normal DRAMA functions required to implement rescheduling actions are provided as Tcl commands. For example, `DitsPutObeyHandler`to change the handler for the next reschedule event, `DitsPutKickHandler` to change the handler for the next Kick message and `DitsPutRequest` to set the reschedule request. Note that there is no `DitsPutDelay`, you can use an argument to `DitsPutRequest` to set reschedule timeouts.

You should note that if you use the original Dtcl message commands (obey, kick etc.) in these actions, any responses are returned to the user interface and the action will not be invoked again. To invoke the action again for replies, the normal DRAMA technique, send the messages with the Tcl commands `DitsMessage`, `DitsRequestNotify` and `DitsLoad`.

There are also C level calls equivalent to the Tcl commands `DitsPutAction`, `DitsPutObeyHandler()` and `DitsPutKickHandler()`. This allow you to specify from C level, Tcl commands to be invoked to implement actions. The C calls are `DtclPutAction()`, `DtclPutTclObeyHandler()` and `DtclPutTclKickHandler`. You can change between C and Tcl handlers thoughout your action as long as your first handler is Tcl.

# 4 Graphical User Interfaces

The command `dtk` runs a version of the **DTCL** task which incorporates Tk and is used to create a graphical user interface rather than a command line one. The command options are exactly the same as those of `dtcl`. The -t option should specify a Tcl script which contains the Tk commands to create the desired graphical user interface. User interface events can then be used to send messages to other **DRAMA** tasks in the system via the `obey` and similar commands.

# 5 The DTCL Package

The **DTCL** package provides a callable interface which allows a command line or graphical user interface to be incorporated into any DRAMA task. In fact the **DTCL** task itself is simply a task which activates the **DTCL** package and has no other actions and parameters.

## 5.1 Command Line User Interfaces

The following example shows how the **DTCL** package is incorporated into a task. The main requirement is that the routine `DtclAppMainLoop` is called in place of `DitsMainLoop` to set up the user interface.

```
DitsAppInit(thisTask,bufsize,DITS_M_X_COMPATIBLE,0,&status);
SdpInit(&parid,&status);
DtclAppMainLoop(argc,argv,MessageBytes,MaxMessages,ReplyBytes,MaxReplies,
     tclScript,1,"Dtcl> ",DtclCommands,0,&status);

return(DitsStop(thisTask,&status));
```

There are a few important points to note:

- DitsAppInit must be called with the `DITS_M_X_COMPATIBLE` flag.
- **DTCL** requires a parameter system which uses SDS to store the parameters (such as the SDP parameter system).
- `argc` and `argv` from the C `main()` function are passed to `DitsAppMainLoop`. This means that all applications support some of the options to the `Dtcl` command, such as `-m`.

The argument `DtclCommands` is a routine which will be called after the Tcl interpreter has been created, and is normally used to add any Tcl commands which are specific to the application. It would do this by calling the Tcl routine `Tcl_CreateCommand`. Below is a simple example of how this routine would be written.

```
int DtclTest(ClientData data, Tcl_Interp *interp, int argc, char *argv[])

{
   printf("Command test executed\n");
   return TCL_OK;
}


void DtclCommands(Tcl_Interp *interp, void * clientData, StatusType *status)


{
   Tcl_CreateCommand(interp,"test",DtclTest,(ClientData *)NULL,
                (Tcl_CmdDeleteProc *)NULL);
}
```

Set DtclCommands to 0 if it is not requied.

### 5.2 Graphical User Interfaces

To create a Tk graphical user interface is very similar but the routine `DtclAppTkMainLoop` is used in place of `DtclAppMainLoop`. The tclScript parameter must specify a Tcl script which sets up the graphical user interface

```
DitsAppInit(thisTask,bufsize,DITS_M_X_COMPATIBLE,0,&status);
SdpInit(&parid,&status);
DtclAppTkMainLoop(argc,argv,MessageBytes,MaxMessages,ReplyBytes,
     MaxReplies,tclScript,1,"Dtcl> ",DtkCommands,0,&status);

return(DitsStop(thisTask,&status));
```

### 5.3 Dtcl_Init

From **DRAMA** version 1.0, assuming it was built with Tcl version 7.5 or later, it is possible to add **DRAMA** to a Tcl/Tk program using the standard Tcl technique (`Tcl_AppInit`. **DRAMA** now provides a `Dits_Init` function. When invoked from a standard Tcl program initailisaction sequence, this makes the program into a DRAMA task and adds all the DTCL commands.

Since we are adding DRAMA to an existing program, the behaviour is a bit different. The prompt, if any, is no longer dependent on the parameter `DTCL_PROMPT`, which no longer exists, but is the standard Tcl/Tk prompt. The parameter `DTCL_ENABLE` is not relevant and also has been removed. In addition, the various options to the DRAMA initailisaction must be provided using Tcl variables. The following are used

**DramaTaskName** Sets the name of the task. If not set, then it defaults to "`[winfo name]`" when executing within an application with Tk available or "`[file tail [info nameofexecutable]]`" otherwise.

**DramaGlobalBufSize** Sets the task global buffer size. The default value is 150000 bytes.

**DramaSelfBufSize** Sets the task self buffer size. The default value is 2000 bytes.

See the C routine `DitsAppInit()` for full details of the meanings of these items.

If you use this routine, you should note that by default, the standard Tcl program (`tclsh`) does not enable use of its event loop (although wish and its derivites do). With no event loop, you won't be able to receive DRAMA messages. To enable the event loop, you must invoke an appropriate Tcl command, such as `vwait` or the DRAMA specific command `DramaEventLoop`. The former command does not accept input from standard output whilst the `DramaEventLoop` does. `DramaEventLoop` takes one optional argument. If supplied, it is the name of a variable the setting of which causes the command to return. If not suppliied, the command will never return (although it always appears to return immediately since it isssues a new prompt).

### 5.4 Linking

In order to link a task that uses `DtclAppMainLoop` the link command must include the following in addition to the standard **DRAMA** libraries:

```
-ltcl -lm
```

To use `DtclAppTkMainLoop` the following is needed:

```
-ltcl -ltk -lX11 -lm
```

If these link commands fail to find the Tcl and Tk libraries it may be that Tcl and Tk are not installed on the system you are using. Tcl and Tk are not included in the **DRAMA** release and must be installed seperately. The current version of DTCL has been tested using Tcl version 7.6/Tk version 4.2 and Tcl/Tk 8.0. It should be compatible with any versions since Tcl 7.0 and Tk 3.3. The home site for Tcl is `http://www.sunlabs.com/research/tcl`. It is also possible that Tcl is installed, but is either not in the standard location or the libraries have slightly different names (`libtcl42.a` for example instead of just `libtcl.a`).

## 6 Loading Dtcl dynamically

From `Tcl 7.4/Tk 4.0`, is was possible on many platforms to dynamically load Tcl/Tk extensions into an interpreter. This is possible in DRAMA from DRAMA version 1.0 although only if DRAMA was built with Tcl version 7.5 or later (since some other features required are only available from version 7.5). Not all platforms are supported as yet, but this is changing quickly, with SunOs, Solaris and Digital Unix (OSF) supported.

You need to load into Tcl the module-

```
$DTCL_DEV/libdtcl.so
```

This can be done using the Tcl load command directly, but this is not the standard approach.

You should consider adding the following to your pkgIndex.tcl file

```
package ifneeded Dtcl 1.0 {
    global env
    if {[lsearch -exact [array names env] DTCL_DEV] != -1} {
        load $env(DTCL_DEV)/libdtcl.so
    } else {
 error "DTCL_DEV environment variable not defined, have you started up drama?"
    }
}
```

Normally, pkgIndex.tcl is in the same directory as your Tcl library, often `/usr/local/lib`. Once this is done, then the Tcl command

```
package require Dtcl
```

will work, as will the other package commands.

As might be expected, this facility uses the `Dtcl_Init` routine described above. See that routine for details of options.

# 7 Usage notes

There are some items which don't fit in the the above discussion but which should be bought out.

## 7.1 Sychronous/Asynchronous interaction problems

Some problems have been found in the interactions between Sychronous and Asynchronous code.

### 7.1.1 The `-wait` Option and `after`

Each of the DTCL message sending commands (obey, kick etc.) support a `-wait` option, described in section 3.2. One potential problem which can occur with the use of this option within the callback procedure of another command. For example, say you have started an obey using the following

```
obey MYTASK MYACTION -success succ_proc
```

Now you may want the success handler to start another action. The following will fail

```
proc succ_proc { id } {
     obey MYTASK ACTION2 -wait
}
```

This is because the use of -wait causes a recursive call to the DRAMA message loop, which is not yet supported. In older versions of Dtcl, this caused a major failure. In later versions you should just get an error message. It is hoped a future version of Dtcl will support this.

There is nothing here to stop you doing a sequence of commands using `-wait`, or from doing commands using `-wait` in Tk widget callbacks. The only restriction is where the callback is a DRAMA callback.

Note that this problem also applies to the use of the `-wait` and `-waitload` options of the `DramaLoad` and `RequestNotify` commands and to the `after` command. In the later case, you should treat the `after` command without a callback in the same way as the use of `-wait` and its use with a callback the same as without `-wait` - e.g. you cannot use `-wait` in the callback of an `after` command etc.

### 7.1.2 `tkwait`, `vwait`, `update` and `tk_dialog`

For similar reasons to the problem with the use of `-wait` in callback procedures, the commands `tkwait`, `vwait` and `update` can cause problems. The former commands should not be used at all in DRAMA triggered callbacks (including `after` command callbacks). The `update` command can be used but only if the option `idletasks` is supplied.

These restrictions have implications on the use of the `tk_dialog` command, since it uses these commands. An alternative procedure - `DtclTkDialog` is provided and documented in appendix C. It provides a callback instead of waiting for buttons to be pressed.

Since the default version of the `tkerror` procedure uses `tk_dialog`, you should redefine `tkerror` to call the equivalent routine provided by DTK - `DtclTkError`.

You should also consider any other Tcl/Tk supplied dialogs, in particular, any command which creates a dialog and blocks waiting for the result. Most such commands will use something like `vwait` or `tkwait`. Consider modifying such scripts to invoke a command when the event occurs instead of blocking.

## 7.2 Background Errors

Since the callback procedures to the various messaging commands (obey, kick, after etc.) are invoked asychronously, there is no place for the interperter to return error messages. Note here that I am talking about errors in the interpting of the script or as a result of invoking the `error` command, not message failures which cause will the error callback (if any) to to invoked.

Dtcl uses a similar approach to Tk in that a special command is invoked to report the message. This command is `DtclError` and is described in Appendix C. The Tk version of Dtcl provides a default implementation of this command which puts up a dialog box and provides an option to view a stack dump, in a similar way to what happens if a Tk background error occurs.

### 7.2.1 C level handling of Background Errors

If you write Asynchronous C code which needs to output background errors, you can use the same technqiue used by DRAMA. Use code like the following

```
result = Tcl_GlobalEval(interp,callback_command);
if (result != TCL_OK)
{
     Tcl_AddErrorInfo(interp,"Background Error occurred in ???");
     DtclBackgroundError(interp);
}
else
{
    ....
```

**DtclBackgroundError()** is documented in appendix B. (As an aside, note the use of `Tcl_GlobalEval` instead of alternatives. A callback would normally be executed at global level).

### 7.3 Reporting script errors to DRAMA

If Tcl commands are being executed as part of DRAMA actions routines etc., you will probably want to report any Tcl error using the DRAMA error reporting system. The routine `DtclErsRep` can be used. This routine examines a Tcl interperter error messages and reports the error using `ErsRep`. The following bit of code is from a DRAMA action which invokes `Tcl_GlobalEval`.

```
/* Setup and execute a Tcl command. */
int result;
char buffer[20];
Tcl_DString command;
Tcl_DStringInit(&command);
Tcl_DStringAppend(&command,"Message",-1);
Tcl_DStringAppendElement(&command,string);
sprintf(buffer,"%d",DitsGetActIndex());
Tcl_DStringAppendElement(&command,buffer);
result = Tcl_GlobalEval(MyInterp,command.string);
Tcl_DStringFree(&command);
/* Did an error occur in executing the Tcl command */
if (result != TCL_OK)
{
    /* Put the result in our return argument structure */
    SdsIdType id;
    ArgNew(&id,status);
    ArgPutString(id,"Argument1",MyInterp->result,status);
    DitsPutArgument(id,DITS_ARG_DELETE,status);
    /* Report the Tcl error using Ers */
    *status = DTCL__TCLCMD;
    DtclErsRep(MyInterp,1,status);
    /* The error will be reported when the action completes */
}
```

The second argument to `DtclErsRep` is a flag which should be set true if you want the Tcl stack dump and value of `errorInfo` reported. Otherwise, just the plain error message reported. In either case, each line of the error message is passed to `ErsRep`, resulting in a reasonable error report format.

## 7.4 Errors from DTCL's Tcl commands

DTCL sticks to a standard way of reporting errors from Tcl commands. Errors are returned by setting a Tcl result string to the command name followed by a descriptive text string, suitable for display in interactive suitations.

Since the underlying DRAMA routines generally report errors by setting a status value to a predefined value, it is useful to be able to get at this value. DTCL supports this using the Tcl `errorCode` variable. On all errors generated by DTCL, this is set to a 5 element list with items as follows

1. Always "DTCL". This enables your code to determine the error is in the DTCL format and the rest of `errorCode` will be as follows
2. The name of facility which originated the error.
3. The severity of the message. See the MESSGEN system for meanings of severities.
4. The message identification name assocaited with the status code which caused the error to be generated
5. An integer which is the status code which causes the error to be generated.

Often the error text return by the command will be generated from the status code, but sometimes this is not appropiate and the text is generated by the command. This means you may have different error text values for the same error code.

When writting new Tcl procedures using DRAMA routines, you should use the routine `DtclCmdError` to generate errors in the same way.

## 7.5 Floating point precision in new Tcl commands

In section 5.7 of the Tcl book [2], the use of the variable `tcl_precision` to set the precision of floating point calculations is described. If you are writing new Tcl commands which return floating point values in strings, you should make use of this variable to determine the required precision. This makes your code consistent with Tcl itself. As an example, take a Tcl command which returns the value of pi.

```
int Pi(ClientData clientData, Tcl_Interp *interp,
                                int argc, char *argv[])

#   define PI 3.141592654
```

```
    int precision = 6;  /* 6 is the Tcl default */
    char *value;
    char format[10];

    if ((value = Tcl_GetVar(interp, "tcl_precision",
                            TCL_GLOBAL_ONLY)) != NULL)
    {
        if (Tcl_GetInt(interp, value, &precision ) == TCL_ERROR)
        {
            Tcl_AddErrorInfo(interp,
"Error reading value of \"tcl_precision\" variable as in integer");
/*
 *     Use DtclCmdError to setup a Dtcl error return with status code
 *     MYFAC__PRECISION.
 */
            return DtclCmdError(interp,argv[0],MYFAC__PRECISION);
        }
    }
/*
 *  Create a format statment specifying the correct precision
 */
    sprintf(format,"%%.%dg",precision);
/*
 *  Output the result
 */
    sprintf(interp->result,format,PI);
    return TCL_OK;
}
```

It should be noted that this approach changes from Tcl version 8.0. From that version it is possible to return the result as a floating point value, meaning you don't need to worry about `tcl_precision`. Read the Tcl version 8.0 man pages and release notes for details.

## References

[1] Tony Farrell, AAO. *21-Dec-1992, Guide to Writing Drama Tasks.* Anglo-Australian Observatory **DRAMA** Software Document.

[2] John K. Ousterhout, *Tcl and the Tk Tookkit*, Addison-Wesley 1994.

# A The dtcl and dtk commands

## A.1 dtcl — TCL user interface to DRAMA

**Function:** TCL user interface to DRAMA

**Description:** This program provides a command line user interface to DRAMA based on the Tcl command language.

It extends Tcl to provide commands to send obey, kick and parameter set and get messages to tasks, and to allow building of SDS argument structures.

DTCL is itself a DRAMA task and has an action DTCL_COMMAND which allows a command to be invoked in its Tcl interpreter.

**Synopsis:** dtcl [options]

**Command options:**

**-b size** Size is the total message buffer size. Default 15000. It is automatically increased such that at least one reply buffer of the size specified by -m will be accepted. You will only have to use this option if you will be communicating with more then one task.

**-m n1:n2:n3** Sets message buffer sizes for connections to other tasks. n1 = Messages bytes allowed for message to be sent n2 = Messages bytes allowed for return messages n3 = Number of return messages which may occur Default is 8000:800:10 which should be sufficient for most tasks

**-n name** The name this task should register itself as. Default is DTCL. This option may be required if you intend to have several versions of this program outstanding at one time.

**-t file** Filename of a Tcl script to be invoked on intialization.

**-logging** Enable logging of DRAMA messages.

**-mayload** This program may load other programs directly. Normally, the DRAMA master task must be running.

**-nostdin** Don't prompt for commands from stdin.

**-globalnames** Force DRAMA names to be placed in the global namespace. For Tcl 8.0+ only. For prior versions of TCL, DRAMA names are always placed in the global namespace.

**-noglobalnames** Force DRAMA names to not be placed in the global namespace. For Tcl 8.0+ only. For prior versions of TCL, DRAMA names are always placed in the global namespace.

**Language:** C

**See Also:** DTCL manual, dtk(1), Dtcl_Init(3), tclsh(1), DtclAppMainLoop(3), DitsAppInit(3), Dits Specification.

**Support:** Jeremy Bailey, AAO

---

## A.2 dtk — Tk user interface to DRAMA

**Function:** Tk user interface to DRAMA

**Description:** This program is used to set up graphical user interface to DRAMA based on the Tcl command language and Tk widget set.

The -t option should be used to specify a Tcl script which defines the graphical user interface. The user interface events can be used to invoke actions in Drama tasks by sending messages to them.

It extends Tcl to provide commands to send obey, kick and parameter set and get messages to tasks, and to allow building of SDS argument structures.

DTK is itself a DRAMA task and has an action DTCL_COMMAND which allows a command to be invoked in its Tcl interpreter.

**Synopsis:** dtk [options]

**Command options:**

**-b size** Size is the total message buffer size. Default 15000. It is automatically increased such that at least one reply buffer of the size specified by -m will be accepted. You will only have to use this option if you will be communicating with more then one task.

**-m n1:n2:n3** Sets message buffer sizes for connections to other tasks. n1 = Messages bytes allowed for message to be sent n2 = Messages bytes allowed for return messages n3 = Number of return messages which may occur Default is 8000:800:10 which should be sufficient for most tasks

**-n name** The name this task should register itself as. Default is DTCL. This option may be required if you intend to have several versions of this program outstanding at one time.

**-t file** Filename of a Tcl script to be invoked on intialization.

**-logging** Enable logging of DRAMA messages

**-mayload** This program may load other programs directly. Normally, the DRAMA master task must be running.

**-display name** Specify X display

**-geometry value** Specify initial window geometry.

**-sync** Run with X Synchronize enabled

**-stdin** If a script is specified, accept commands from stdin after reading the script. By default, we don't use stdin if a script is specified.

**-globalnames** Force `DRAMA` names to be placed in the global namespace. For Tcl 8.0+ only. For prior versions of `TCL`, `DRAMA` names are always placed in the global namespace.

**-noglobalnames** Force `DRAMA` names to not be placed in the global namespace. For Tcl 8.0+ only. For prior versions of `TCL`, `DRAMA` names are always placed in the global namespace.

**Language:** C

**See Also:** `DTCL` manual, dtcl(1), Dtcl_Init(3), wish(1), DtclTkAppMainLoop(3), DitsAppInit(3), Dits Specification.

**Support:** Jeremy Bailey, `AAO`

**Limitations:** Currently not avaliable under VxWorks

---

# B The DTCL routines

## B.1 DtclAppMainLoop — Provide a TCL user interface in a DRAMA task.

**Function:** Provide a TCL user interface in a DRAMA task.

**Description:** This routine can be used to give any Drama task a command line user interface based on Tcl. It should be called in place of DitsMainLoop.

For the user interface to work, DitsAppInit must have been called with the DITS_M_X_COMPATIBLE flag.

If argc is positive, then the following Tcl global variables are set

**argv0** is set to argv[0]

**argc** is set to argc

**argv** is set to the argv[] (element 1 onwards)

DtclAppMainLoop adds the following actions and parameters to the task:

| | |
|---|---|
| DTCL_COMMAND | An action which causes a Tcl Command to be executed in the user interface's Tcl interpreter. It should have a single character string argument which is the Tcl command to be executed. It enables another task to issue any command which could be typed directly at the user interface. |
| DTCL_ENABLE | An integer parameter which is 1 if the user interface is currently enabled and 0 if it is disabled. |
| DTCL_PROMPT | A character string parameter which specifies the current prompt string for the user interface. |

**Options:** The following options specfied in argv are used

**-m n1:n2:n3** Overrides the message buffer sizes for connections to other tasks.

| | |
|---|---|
| n1 | Message bytes allowed for messages to be sent. |
| n2 | Message bytes allowed for return messages |
| n3 | Number of return messages which may occur. |

**-logging** Enable logging of DRAMA messages

**-nostdin** Don't prompt for commands from stdin.

**-globalnames** Force DRAMA names to be placed in the global namespace. For Tcl 8.0+ only. For prior versions of TCL, DRAMA names are always placed in the global namespace.

**-noglobalnames** Force DRAMA names to not be placed in the global namespace. For Tcl 8.0+ only. For prior versions of TCL, DRAMA names are always placed in the global namespace.

**Call:**

DtclAppMainLoop(argc, argv, MessageBytes, MaxMessages, ReplyBytes, MaxReplies, TclScript, enable, prompt, CommandDefn, clientData, status);

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(>) argc (int)** Argc as passed to the main function.

**(>) argv (char **)** Argv as passed to the main function.

**(>) MessageBytes (int)** Bytes to allocate to send message.

**(>) MaxMessages (int)** Maximum number of messages to send.

**(>) ReplyBytes (int)** Bytes to allocate for replies.

**(>) MaxReplies (int)** Maximum number of replies.

**(>) TclScript (char *)** File name of a Tcl script to execute on initialization.

**(>) enable (int)** 1 to enable the user interface, 0 to disable it.

**(>) prompt (char *)** Prompt string for the user interface.

**(>) CommandDefn (DtclAppCommandDefnType)** A routine which will be called after creation of the Tcl interpreter which can be used to define application specific Tcl commands. If zero, not used.

**(>) clientData (void *)** Item passed to CommandDef routine

**(!) status (StatusType *)** Modified status.

**Language:** C

**Support:** Jeremy Bailey, `AAO`

**Prior Requirements:** DitsAppInit and SdpInit must have been called. The `DITS_M_X_COMPATIBLE` flag must have been specified to DitsAppInit.

**See Also:** `DTCL` manual, dtcl(1), Dtcl_Init(3), tclsh(1), DtclTkAppMainLoop(3), DitsAppInit(3), SdpInit(3), Dits Specification.

---

## B.2 DtclAppTkMainLoop — Provide a Tk user interface in a `DRAMA` task

**Function:** Provide a Tk user interface in a `DRAMA` task

**Description:** This routine can be used to give any Drama task a graphical user interface based on Tcl and Tk. It is called in place of DitsMainLoop.

For the user interface to work, DitsAppInit must have been called with the `DITS_M_X_COMPATIBLE` flag.

DtclTkActivate adds the following actions and parameters to the task:

| | |
|---|---|
| DTCL_COMMAND | An action which causes a Tcl Command to be executed in the user interface's Tcl interpreter. It should have a single character string argument which is the Tcl command to be executed. It enables another task to issue any command which could be typed directly at the user interface. |

The following Tcl global variables are set

**tcl_interactive** is set true is stdin is a tty, else set false.

**logging** Set to the value of the -logging flag

**hostname** Is set to the name of the host we are running on

If argc is positive, then the following Tcl global variables are set

**argv0** is set to argv[0]

**argc** is set to argc

**argv** is set to the argv[] (element 1 onwards)

**Options:** The following options specfied in argv are used

**-m n1:n2:n3** Overrides the message buffer sizes for connections to other tasks. n1 = Message bytes allowed for messages to be sent. n2 = Message bytes allowed for return messages n3 = Number of return messages which may occur.

**-logging** Enable logging of DRAMA messages

**-display name** Specifies the X display to be used for the main window.

**-geometry string** Specifies the default geometry string.

**-sync** Run with X Synchronize enabled

**-stdin** If a script is specified, accept commands from stdin after reading the script. By default, we don't use stdin if a script is specified.

**-globalnames** Force DRAMA names to be placed in the global namespace. For Tcl 8.0+ only. For prior versions of TCL, DRAMA names are always placed in the global namespace.

**-noglobalnames** Force DRAMA names to not be placed in the global namespace. For Tcl 8.0+ only. For prior versions of TCL, DRAMA names are always placed in the global namespace.

**Call:**

DtclAppTkMainLoop(argc, argv, MessageBytes, MaxMessages, ReplyBytes, MaxReplies, TclScript, enable, CommandDefn, clientData, status);

**Parameters:** ("**>**" input, "**!**" modified, "W" workspace, "**<**" output)

**(>) argc (int)** Argc as passed to the main function.

**(>) argv (char **)** Argv as passed to the main function.

(>) **MessageBytes (int)** Bytes to allocate to send message.

(>) **MaxMessages (int)** Maximum number of messages to send.

(>) **ReplyBytes (int)** Bytes to allocate for replies.

(>) **MaxReplies (int)** Maximum number of replies.

(>) **TclScript (const char *)** File name of a Tcl script to execute on initialization.

(>) **enable (int)** 1 to enable the user interface, 0 to disable it.

(>) **CommandDefn (DtclAppCommandDefnType)** A routine which will be called after creation of the Tcl interpreter which can be used to define application specific Tcl commands. If zero, not used.

(>) **clientData (void *)** Item passed to CommandDef routine

(!) **status (StatusType *)** Modified status.

**Language:** C

**Support:** Jeremy Bailey, AAO

**Prior Requirements:** DitsAppInit and SdpInit must have been called. The DITS_M_X_COMPATIBLE flag must have been specified to DitsAppInit.

**See Also:** DTCL manual, dtk(1), Dtcl_Init(3), wish(1), DtclAppMainLoop(3), DitsAppInit(3), SdpInit(3), Dits Specification.

---

## B.3 DtclBackgroundError — Handle errors in Tcl commands invoked in Dtcl callbacks

**Function:** Handle errors in Tcl commands invoked in Dtcl callbacks

**Descriptions:** This procedure is invoked to handle errors that occur in Tcl commands that are invoked in "background" - e.g. From Dtcl callbacks.

The Tcl command "DtclError" is invoked to process the error, passing it the error message. If that fails, then an error message is output using ErsOut.

This routine derived from tk Tk_BackgroundError

**Include File:** dtcl.h

**Call:**
DtclBackgroundError(interp)

**Parameters:** (">" input, "!" modified, "W" workspace, "<" output)

(!) **interp (Tcl_Interp)** A Tcl interperter.

**See Also:** DTCL manual, DtclError(n)

**Author:** Tony Farrell, AAO

---

## B.4 DtclCmdError — Handle errors in Dtcl Tcl command implementations.

**Function:** Handle errors in Dtcl Tcl command implementations.

**Descriptions:** This procedure is invoked to handle errors that occur in Dtcl's Tcl commands. It sets the error string, and code in and appropiate and consistent maner.

There are two possibilities. First, if the call is indicating an error caused by a Drama status variable being set bad. In that case the user calls this routine with the name of the command and the bad status. The result string and errorCode will be set. ErsFlush() will be invoked to flush reported errors.

The second case is where the caller has already formated a result into the return value. All we need to do is set the errorCode.

In both cases, the error code is set to a list containing the following values

| | |
|---|---|
| Marker_String | Will contain "DTCL", which indicates errorCode contains a value generated by this routine. |
| Facility | The facility associated with the error code |
| Severity | The severity associated with the error code |
| Name | The message name assocaited with the error code |
| Value | The value of the message code as an integer. |

**Include File:** dtcl.h

**Call:**

(int) = DtclCmdError(interp,cmd,error)

**Parameters:** ("**>**" input, "**!**" modified, "W" workspace, "**<**" output)

**(!) interp (Tcl_Interp)** A Tcl interperter.

**(>) cmd (char *)** Name of the command. If null, then it is assumed the interpeter's result string has already been set to the appropriate value and all we have to do is set the errorCode.

**(>) error (StatusType)** The status assocaited with the error.

**Returned Value:** Always TCL_ERROR;

**See Also:** DTCL manual, error(n)

**Author:** Tony Farrell, AAO

### B.5 DtclErsRep — Report Tcl errors using Ers functions.

**Function:** Report Tcl errors using Ers functions.

**Descriptions:** This procedure can be used to report Tcl script errors using the `DRAMA` ErsRep function. It is generally used to handle errors returned from Tcl_Eval and associated functions executed by `DRAMA` actions.

This routine takes the Tcl interperter and reports the last error which occured in that interperter using a sequence of ErsRep calls such that one ErsRep calls occurs for each line in the error report. An optional flag (stack) indicates if the contents of the errorInfo and errorCode variables should be reported .

It is left up to the user to flush the errors.

**Include File:** dtcl.h

**Call:**

DtclErsRep(interp,stack,status)

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(>) interp (Tcl_Interp)** The Tcl interperter to report for.

**(>) stack (int)** If true, report the stack dump (errorInfo variable) and errorCode as well as the actual error.

**(!) status (StatusType *)** Modified Status. Passed directly ErsRep which means you should set it before entry for use in the calls to ErsRep. Will only be changed if ErsRep changes it.

**Author:** Tony Farrell, `AAO`

**See Also:** `DTCL` manual, Ers manual, Dits specification.

---

### B.6 DtclPutAction — Add a new action to this task.

**Function:** Add a new action to this task.

**Description:** Adds a new action to the current task. When the action is invoked, the specified Tcl command will be invoked.

To reschedule from Tcl level, use DitsPutRequest(n) and use DitsPutObeyHandler(n) and DitsPutKickHandler(n) to change the handler commands.

To reschdule from C level, use the normal DitsPutObeyHandler(3)/ DitsPutKickHandler(3)/ DitsPutRequest(3) C routines or if you wish from a C routine to specify Tcl routines for the next invocation of an action use DtclPutTclObeyHandler(3)/DtclPutTclKickHandler(3). The use of these two routines rely on you having created this action with either this command or the Tcl level call DitsPutAction(n).

This routine should only be invoked at `UFACE` context. The removal of the malloced data structures used to support this feature relies on you `NOT` calling DitsPutCode(3).

**Language:** C

**Call:**
(void) = DtclPutAction (interp,name,obeyhandler,kickhandler,flags, checkhandler,status)

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(>) interp (Tcl_Interp *)** The Tcl interperter to used when exeucting the commands.

**(>) name (const char *)** The name of the new action. Should not exceed `DITS_C_NAMELEN` in length;

**(>) obeyhandler (const char *)** The Tcl command to execute as the obey handler.

**(>) kickhandler (const char *)** The Tcl command to execute as the kick handler.

**(>) flags (int)** The flags for the action. See the flags part of the structure used by DitsPutActions.

**(>) checkhandler (const char *)** A Tcl command to execute as the spawn check handler. Returns successfully if yes, returns the appropiate error if not.

**Include files:** dtcl.h

**External functions used:**

**Prior requirements:** Can only be executed from a routine which is `NOT` a `DRAMA` action handler routine (should be `UFACE` context code).

**See Also:** `DTCL` manual, DitsPutRequest(n), DitsPutRequest(3), DitsPutObeyHandler(n), DitsPutObeyHandler(3), DitsPutKickHandler(n), DitsPutKickHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DtclPutTclObeyHandler(n), DtclPutTclKickHandler(3), DitsPutCode(3), DitsPutAction(n), DitsPutActions(3).

**Support:** Tony Farrell, `AAO`

---

## B.7 DtclPutTclKickHandler — Sets a new Tcl command to handle next kick messages for this action.

**Function:** Sets a new Tcl command to handle next kick messages for this action.

**Description:** Called by an Action routine to set a Tcl command to be invoked on the next kick of this action.

This routine should only be invoked from actions which where set up using the Tcl command DitsPutAction or the C routine call DtclPutAction.

**Language:** C

**Call:**
(void) = DtclPutTclKickHandler (command,status)

**Parameters:** ("**>**" input, "**!**" modified, "W" workspace, "**<**" output)

**(>) command (char *)** The new Tcl command to be invoked on the next kick of this action.

**Include files:** dtcl.h

**External functions used:**

**External values used:**

**Prior requirements:** Should only be invoked from actions which where set up using the Tcl command DitsPutAction or the C routine call DtclPutAction.

**See Also:** DTCL manual, DitsPutRequest(n), DitsPutRequest(3), DitsPutObeyHandler(n), DitsPutObeyHandler(3), DitsPutKickHandler(n), DitsPutKickHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DtclPutTclObeyHandler(n), DitsPutAction(n), DtclPutAction(3).

**Support:** Tony Farrell, AAO

---

## B.8 DtclPutTclObeyHandler — Sets a new Tcl command for the next reschedule of this action.

**Function:** Sets a new Tcl command for the next reschedule of this action.

**Description:** Called by an Action routine to set a the Tcl command to be invoked on the next reschedule of this action.

This routine should only be invoked from actions which where set up using the Tcl command DitsPutAction or the C routine call DtclPutAction.

**Language:** C

**Call:**
(void) = DtclPutTclObeyHandler (command,status)

**Parameters:** ("**>**" input, "**!**" modified, "W" workspace, "**<**" output)

**(>) command (char *)** The new Tcl command to be invoked on the next reschdule of this action.

**Include files:** dtcl.h

**External functions used:**

**External values used:**

**Prior requirements:** Should only be invoked from actions which where set up using the Tcl command DitsPutAction or the C routine call DtclPutAction.

**See Also:** DTCL manual, DitsPutRequest(n), DitsPutRequest(3), DitsPutObeyHandler(n), DitsPutObeyHandler(3), DitsPutKickHandler(n), DitsPutKickHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DtclPutTclKickHandler(3), DitsPutAction(n), DtclPutAction(3).

**Support:** Tony Farrell, AAO

---

## B.9 DtkCommands.c — Tcl command adding procedure for Dtk.

**Function:** Tcl command adding procedure for Dtk.

**Description:** This procedure is invoked by dtk to add extra Dtk commands. The default implementation is null, other implementations can be used to easily extend dtk. Note, you should also set the default task name via the variable DtkName. To link in this module, specify $$DTCL_LIB/dtk.o to your linker command before you specify the Dtcl library.

Consider using the Dtcl_Init procedure instead of this approach if the Tcl version certain to be Tcl 7.5 or later.

**Language:** C

**Call:**
(void) = DtkCommands (interp,clientData,status)

**Parameters:** ("**>**" input, "**!**" modified, "W" workspace, "**<**" output)

**(!) interp (Tcl_Interp *)** The Tcl interperter

**(>) clientData (void *)** Pointer to user specified data.

**(!) status (StatusType *)** modified status.

**Include files:** None.

**External functions used:** None.

**External values used:** None.

**Prior requirements:** None.

**See Also:** DTCL manual, Dtcl_Init(3), DtclTkAppMainLoop(3), dtk(1), wish(1).

**Support:** Tony Farrell, AAO

---

### B.10 Dtcl___SharedInit — Add The DRAMA Tcl interface to the specified interpreter.

**Function:** Add The DRAMA Tcl interface to the specified interpreter.

**Description:** The full DRAMA Tcl interface is added to the interpreter specified. It is not yet clear what is the effect of adding DRAMA to a restricted interpreter. The version of the package will be that specified by the DRAMA_VERSION macro as defined in drama.h when the module was compiled. The package name is "Dtcl".

This command is not available in TCL versions prior to 7.5. In those cases it returns TCL_ERROR.

Use of TCL Namespaces is the default from version 8.0 of TCL, with all DRAMA commands and variables placed in the "drama" namespace. There are two ways of changing this default. First, a system build time option and secondly the environment variable DTCL_NAMESPACES. If set to 0, the global namespace is used. If set to non-zero, DRAMA uses namespaces. Which ever default is enabled, the value can be overrided at run time by setting the global TCL variable DramaForceGlobal. If this has a positive value, the global namespace is used. If it has the value 0, the drama namespace is used. It is is negative, the default value is used. Note, this variable is always a global variable, regardless of the namespace mode.

Note that under TCL 8.0 and later, if use of the global namespace is enabled, DRAMA commands are placed in the drama namespace but imported into the global namespace. Likewise, drama variables are placed in the drama namespace but upvar is used to link them to global names.

For all variables mentioned below, if namespace support is enabled, the initial values should be set by using the "namespace eval drama" command to enclose a "variable value" command, where value is the value of the variable, for example

namespace eval drama  variable DramaTaskName MyTask

The DRAMA task name is determined as follows- From the value of the variable DramaTaskName if it exists. Otherwise if tk, then use "winfo name ." Otherwise, use "file tail [info nameofexecutable]".

The DRAMA global buffer size is determined from the variable DramaGlobalBufSize, if it exists. It defaults to 15000 bytes.

The DRAMA self buffer size is determined from the variable DramaSelfBufSize, if it exists. It defaults to 2000 bytes.

The Sdp parameter system is activated but no parameters are added.

The Action DTCL_COMMAND is enabled, which is used as per other Dtcl interfaces.

The caller should note that by default, the standard Tcl program does not enable use of its event loop (although wish and its derivites do). To enable the event loop, you must invoke the Tcl "vwait" command (or a similar command) or the DRAMA specific DramaEventLoop command. The former does not accept input from stdin whilst the later does. DramaEventLoop takes one optional argument. If supplied, it is the name of a variable the setting of which causes the command to return. If not supplied, the command

will never return (althrough it always appears to return immediately since it issues a new prompt).

**Language:** C

**Call:**
(int) = Dtcl___SharedInit (interp)

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(`!`) interp (Tcl_Interp `*`)** The Tcl interpreter

**Return value:** `TCL_OK` or `TCL_ERROR`. The Tcl string will be set to the name the task registered as.

**See Also:** `DTCL` manual, tclsh(1), wish(1), load(n), package(n), dtcl(1), dtk(1), DtclAppMainLoop(3), DtclTkAppMainLoop(3), dtkcommands(3).

**Support:** Tony Farrell, `AAO`

---

## B.11 Dtclb_Init — Add The `DRAMA` Tcl interface to the specified interpreter. (From libdtclb shared

**Function:** Add The `DRAMA` Tcl interface to the specified interpreter. (From libdtclb shared load).

**Description:** The full `DRAMA` Tcl interface is added to the interpreter specified. It is not yet clear what is the effect of adding `DRAMA` to a restricted interpreter. The version of the package will be that specified by the `DRAMA_VERSION` macro as defined in drama.h when the module was compiled. The package name is "Dtcl".

This command is not available in `TCL` versions prior to 7.5. In those cases it returns `TCL_ERROR`.

Use of `TCL` Namespaces is the default from version 8.0 of `TCL`, with all `DRAMA` commands and variables placed in the "drama" namespace. There are two ways of changing this default. First, a system build time option and secondly the environment variable `DTCL_NAMESPACES`. If set to 0, the global namespace is used. If set to non-zero, `DRAMA` uses namespaces. Which ever default is enabled, the value can be overrided at run time by setting the global `TCL` variable DramaForceGlobal. If this has a positive value, the global namespace is used. If it has the value 0, the drama namespace is used. It is is negative, the default value is used. Note, this variable is always a global variable, regardless of the namespace mode.

Note that under `TCL` 8.0 and later, if use of the global namespace is enabled, `DRAMA` commands are placed in the drama namespace but imported into the global namespace. Likewise, drama variables are placed in the drama namespace but upvar is used to link them to global names.

For all variables mentioned below, if namespace support is enabled, the initial values should be set by using the "namespace eval drama" command to enclose a "variable value" command, where value is the value of the variable, for example

namespace eval drama  variable DramaTaskName MyTask

The `DRAMA` task name is determined as follows- From the value of the variable DramaTaskName if it exists, otherwise Otherwise, use "file tail [info nameofexecutable]".

The `DRAMA` global buffer size is determined from the variable DramaGlobalBufSize, if it exists. It defaults to 15000 bytes.

The `DRAMA` self buffer size is determined from the variable DramaSelfBufSize, if it exists. It defaults to 2000 bytes.

The Sdp parameter system is activated but no parameters are added.

The Action `DTCL_COMMAND` is enabled, which is used as per other Dtcl interfaces.

The caller should note that by default, the standard Tcl program does not enable use of its event loop (although wish and its derivites do). To enable the event loop, you must invoke the Tcl "vwait" command (or a similar command) or the `DRAMA` specific DramaEventLoop command. The former does not accept input from stdin whilst the later does. DramaEventLoop takes one optional argument. If supplied, it is the name of a variable the setting of which causes the command to return. If not supplied, the command will never return (althrough it always appears to return immediately since it issues a new prompt).

**Language:** C

**Call:**

(int) = Dtclb_Init (interp)

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(`!`) interp (Tcl_Interp `*`)** The Tcl interpreter

**Return value:** `TCL_OK` or `TCL_ERROR`. The Tcl string will be set to the name the task registered as.

**See Also:** `DTCL` manual, tclsh(1), wish(1), load(n), package(n), dtcl(1), dtk(1), DtclAppMainLoop(3), DtclTkAppMainLoop(3), dtkcommands(3).

**Support:** Tony Farrell, `AAO`

# C Tcl commands added by DTCL

## C.1 ArgDelete — Delete an argument structure

**Function:** Delete an argument structure

**Description:** Delete an SDS structure and free it's identifier.

**Call:**
ArgDelete id

**Parameters:**

**(>) id (int)** SDS identifier of structure to be deleted.

**Language:** Tcl

**Support:** Jeremy Bailey, AAO

**See Also:** DTCL manual, Sds manual, ArgDelete(3), SdsFreeId(n), SdsFreeId(3), SdsDelete(3)

---

## C.2 ArgGet — Get an item from an SDS structure

**Function:** Get an item from an SDS structure

**Description:** A named item is read from an SDS structure and a character string representation of its value is returned.

In Tcl there is only a single version of ArgGet rather than a version for each type as in C — This is because in Tcl all variables are strings.

If the top level item is itself a scalar item of the correct name its value will be returned.

With the exception of string items, the length of the result is limited to TCL_RESULT_SIZE (200 bytes) and you will get an error if the result is longer (Say for arrays). You can use SdsArrayGet to access such items. For string items, the extra space is well defined and hence is handled here. (This has changed with the move to using Tcs_SetResult() rather than writing directly into the result string of the Tcl interpreter, but the main effect is that the limit is now set by the size of a local storage array, which is set to 256 bytes.)

**Call:**
ArgGet id name

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to get.

**Returns:** The value of the SDS object.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgGetString(3), ArgNew(n) ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.3 ArgGetType — Get the type of an Arg style SDS structure

**Function:** Get the type of an Arg style SDS structure

**Description:** Return the type of the item which will be fetched by ArgGet
If the top level item is itself a scalar item of the correct name its type will be returned.

**Call:**
ArgGetType id name

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to get the type of.

**Returns:** The value of the SDS object.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual.

**Support:** Tony Farrell, AAO

---

## C.4 ArgNew — Create a new argument structure

**Function:** Create a new argument structure

**Description:** A new SDS structure is created it and an identifier to it is returned

**Call:**
ArgNew

**Parameters:** None

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgNew(3), SdsNew(3), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.5 ArgPutString — Put a character string item into an SDS structure

**Function:** Put a character string item into an SDS structure

**Description:** A character string item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPutString id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (string)** String to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPutString(3), ArgGet(n), ArgNew(n) ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.6 ArgPutc — Put a character item into an SDS structure

**Function:** Put a character item into an SDS structure

**Description:** A character item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPutc id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (int)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPutc(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.7 ArgPutd — Put a double floating point item into an SDS structure

**Function:** Put a double floating point item into an SDS structure

**Description:** A double floating point item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPutd id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (double)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPutd(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.8 ArgPutf — Put a floating point item into an SDS structure

**Function:** Put a floating point item into an SDS structure

**Description:** A floating point item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPutf id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (float)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPutf(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.9 ArgPuti — Put an integer item into an SDS structure

**Function:** Put an integer item into an SDS structure

**Description:** An integer item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPuti id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (int)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPuti(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.10 ArgPuts — Put a short integer item into an SDS structure

**Function:** Put a short integer item into an SDS structure

**Description:** A short integer item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPuts id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (int)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPuts(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.11 ArgPutu — Put an unsigned integer item into an SDS structure

**Function:** Put an unsigned integer item into an SDS structure

**Description:** A unsigned integer item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPutu id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (int)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPutu(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

## C.12 ArgPutus — Put an unsigned short integer item into an SDS structure

**Function:** Put an unsigned short integer item into an SDS structure

**Description:** An unsigned short integer item is written into a named component within an SDS structure. The component is created if it does not currently exist

**Call:**
ArgPutus id name value

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** Name of component to be written to.

**(>) value (int)** Value to be written.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgPutus(3), ArgGet(n), ArgNew(n), ArgPutString(n) ArgPutx(n)

**Support:** Jeremy Bailey, AAO

---

### C.13 Bell — Ring the terminal bell.

**Function:** Ring the terminal bell.

**Description:** This command rings the terminal bell. If in a dtcl application, this is done by writting a `ASCII` bell character to stderr. In a dtk application, this is done using an appropriate Xlib function.

From Tk 4.2, there is a bell command withing Tk. That command is not available in tcl only based applications so this command is still appropiate in some cases.

**Call:**
Bell

**Parameters:** none

**Language:** Tcl

**See Also:** `DTCL` manual, bell(n)

**Support:** Tony Farrell, `AAO`

---

### C.14 DisconnectHandler — Specify a handler for task Disconnect messages.

**Function:** Specify a handler for task Disconnect messages.

**Description:** This routine specifies the name of a Tcl procedure which will be invoked whenever a task to which this has a pach disconnects.

Note that any handler already put using DitsPutDisConnectHandler will we invoked after the Dtcl handler has been run.

**Call:**
DisconnectHandler [command]

**Parameters:**

**(>) command (string)** A Tcl command to be invoked when a task disconnects The command will be appended with an argument which is a string containing the the name of the task which has disconnected. If not supplied, any current handler is removed.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPutDisconnectHandler(3).

**Support:** Tony Farrell, `AAO`

---

## C.15 DitsActionWait — Blocks the current action and waits for a message to be received.

**Function:** Blocks the current action and waits for a message to be received.

**Description:** The current action is blocked and a message receive loop entered. Other actions can be processed as normal. When a message concerning this action is received, the action is unblocked and will continue. The current entry details (DitsGetArgument(n) DitsGetReason(n) etc) will then be for the first message received for this action. The return value will be the number of other messages still remaining to be processed.

The outstanding messages can be processed by calling this routine repeatly. The action will block if the last call returned a count of 0, except if the -noblock flag is set.

Note, the only reason the return value may be greater then zero is that another action has been invoked and has called this command or the Dits routine of the same name. In that case, the second action must be unblocked and end/reschedule before control will return to the first action.

If messages received while the action was blocked have not been processed when the action completes, they are lost (a message is output to stderr). To ensure these messages are handled, you should continue calling this command until zero is returned.

The action is `NOT` unblocked when a `KICK` is received for the action which invokes it, but the Kick handler may cause the action to be rescheduled and the action will then be unblocked when the reschedule occurs.

This command may be invoked from either Obey or Kick context, but not both at the same time

**Note:** In the current implementation, the Tcl/Tk event loop is not processed whilst waiting for a `DRAMA` message. This restriction will be removed in a future version.

**Call:**
DitsActionWait [options]

**Parameters:** none

**Options:**

**-forget** Forget any remaining messages from previous calls to this routine for this action.

**-noblock** Always return immediatley. We will only read a message if there was one outstanding from a previous call. If there are no outstanding messages or -forget flag has been set, then no message will be read and the return value will be set to -1. Action details (such as DitsGetArgument() etc.) will then be as per prior to the call.

**-nolist** Don't take an entry from the list. If combined with -noblock we simply return the count of outstanding messages. Otherwise we block waiting for a message. When the call returns, it will be as per the first item on the list. It is not clear if this flag is useful to users without the -noblock flag.

**-timeout timeout** A timeout to be applied to the block. Floating point seconds.

**Return value:** Returns the number of messages remaining to be processed. The message which caused the routine to be unblocked is not included. If the -noblock flag was set, then count will be -1 if no messages were available. Count is always non-negative in other cases.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsActionWait(3), DitsUfaceWait(3)

**Prior requirements:** Can only be invoked by Tcl commands executing in the context of a `DRAMA` action.

**Support:** Tony Farrell, `AAO`

---

## C.16 DitsArgument — Fetch the argument of the Dits message which caused this entry

**Function:** Fetch the argument of the Dits message which caused this entry

**Description:** Assuming the current procedure is invoked in the context of a Dits entry, this routine will return the value corresponding to the value of the dits routine DitsGetArgument(3), which will be an Sds id of the argument to the message which caused this entry.

**Call:**
DitsArgument

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetArgument(3), Sds manual.

**Support:** Tony Farrell, `AAO`

---

## C.17 DitsConnectHandler — Specify a handler for task Connect messages.

**Function:** Specify a handler for task Connect messages.

**Description:** This routine specifies the name of a Tcl procedure which will be invoked whenever a task connects to this task.

Note that any handler already put using DitsPutConnectHandler will we invoked after the Dtcl handler has been run.

**Call:**
DitsConnectHandler [command]

**Parameters:**

**(>) command (string)** A Tcl command to be invoked when a task Connects The command will be appended with an argument which is a string containing the the name of the task which has connected. If not supplied, any current handler is removed. At this point this command cannot cause the connection to be rejected or the path details to be modified. If you need to do that, you need to use the C level DitsPutConnectHandler function.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPutConnectHandler(3).

**Support:** Tony Farrell, `AAO`

---

## C.18 DitsDeleteTask — Delete the specified task.

**Function:** Delete the specified task.

**Description:** This routine deletes a task from the system. The interface to this routine allows for the possibility (supported by many systems) that there is a 'polite' way to do this - one that allows the task to find out that it is being deleted, usually through some form of signal handler or exit handler - and a less polite way, which allows the task no chance to intercept the deletion but which is therefore guaranteed to work. The 'Force' option can be set to indicate that the deletion of the process is to be guaranteed; if it is not set then the 'polite' form of deletion, if supported by the system, will be used. Most systems take a certain amount of time to delete a process, so even if a process is successfully deleted it may still show as present for a short time before vanishing from the system, and a calling routine must be prepared for this. If the process to be deleted is on a remote machine, then a task delete request is sent to the `IMP` 'Master' task on that remote machine. If the task was not known to the system, the Known parameter will be returned set false - but this will not cause Status to be returned indicating an error.

If as a result of a call to this routine the target task exits, then any task with outstanding actions to/from that task will be notified.

**Call:**
DitsDeleteTask name [options]

**Parameters:**

**(>) name (string)** The name of the task.

**Options:**

**-force** Task deletion is to be 'forced' as opposed to the default of 'polite'.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsDeleteTask(3).

**Support:** Tony Farrell, `AAO`

---

## C.19 DitsErrorText — Fetch the message string assocaited with a status code.

**Function:** Fetch the message string assocaited with a status code.

**Description:** Returns the message string assocaited with a specified status code.

**Call:**
DitsErrorText code

**Parameters:**

**(>) code (integer)** The status code.

**Options:**

**-notfound** Changes what happens if no text if found for for a message. By default, a standard message is returned containing the code. This flag causes an empty string to be returned if the message text is not found.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsErrorText(3), PutFacility(n), DitsStatusText(n), Mess package.

**Support:** Tony Farrell, `AAO`

---

## C.20 DitsGetActIndex — Return the index of the current action

**Function:** Return the index of the current action

**Description:** Returns the index of the current action. This value should only be used as an argument to DitsSignalByIndex or when creating an argument structure to be used to kick this action, if this action is a spawned action. In that case, the argument item should be named KickByIndex.

**Call:**
DitsGetActIndex

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetActIndex(3), DitsKill(n), DitsSignal(n).

**Support:** Tony Farrell, `AAO`

---

## C.21 DitsGetContext — Returns the Dits context

**Function:** Returns the Dits context

**Description:** Returns the Dits context the caller is being executed within. This is one of

| | |
|---|---|
| `OBEY` | Normal entry caused by either an Obey message or a normal reschedule. |
| `KICK` | Action kicked by reception of a kick message from another task |
| `UFACE` | User interface context, only occurs after a call to DitsUfaceCtxEnable or in a user interface response routine. |

**Call:**

DitsGetContext

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetContext(3).

**Support:** Tony Farrell, `AAO`

---

## C.22 DitsGetEntName — Get the name associated with the current entry of the current action.

**Function:** Get the name associated with the current entry of the current action.

**Description:** This item may or may not be defined for the current entry, depending on the Reason for the entry.

**Call:**

DitsGetEntName

**Parameters:** None

**Language:** Tcl

**See Also:** DTCL manual, DitsGetEntName(3),DitsGetEntInfo(3)

**Support:** Tony Farrell, AAO

---

## C.23 DitsGetEntTransId — Get the transaction id associated with the current entry of

**Function:** Get the transaction id associated with the current entry of the current action.

**Description:** This item may or may not be defined for the current entry, depending on the Reason for the entry.

**Call:**
DitsGetEntTransId

**Parameters:** None

**Language:** Tcl

**See Also:** DTCL manual, DitsGetEntTransId(3),DitsGetEntInfo(3)

**Support:** Tony Farrell, AAO

---

## C.24 DitsGetParId — Get the parameter system id.

**Function:** Get the parameter system id.

**Description:** Gets the parameter system id as returned by the C routine DitsGetParId. Useag of this depend on the parameter system being used by this task but normally this is Sdp, in which case this value is an Sds id refering to the entire parameter system.

**Call:**
DitsGetParId

**Language:** Tcl

**See Also:** DTCL manual, DitsGetParId(3)

**Support:** Tony Farrell, AAO

---

## C.25 DitsGetPath — Gets a path to a task.

**Function:** Gets a path to a task.

**Description:** Initiates a Get Path operation to the specified task. We have to initiate a transaction to get the path, the action which invoked this command will be rescheduled when the response occurs and the id of the transaction initiated is returned.

**Call:**
DitsGetPath taskname [options]

**Parameters:**

**(>) taskname (string)** The name of the task to get a path too. You can specify the node using the task@node format or using the -node option.

**Options:**

**-node node** Specify the node the task is on.

**-buffers buf_sizes** Specify a list of 4 integers being the buffer sizes to use. These will default to the values specified by the Dtcl global buffer variables (MessageBytes/MaxMessages/ReplyBytes/MaxReplies)

**-wait** Wait for the operation to complete, using DitsActionWait to block this action. In this case there is no outstanding transaction id on return so null is returned.

**-waittimeout timeout** If -wait is specified, a timeout to be applied to the wait. Floating point seconds.

**-flowcontrol** Enable flow control on this connection.

**Return value:** Transaction id.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPathGet(3), DitsMessage(n), DitsPutAction(n).

**Prior requirements:** Can only be invoked by Tcl commands executing in the context of a `DRAMA` action. Otherwise the obey/kick/pget/pset/control/monitor/ TaskRunning commands should be used.

**Support:** Tony Farrell, `AAO`

---

## C.26 DitsGetSeq — Returns the sequence count of the current action.

**Function:** Returns the sequence count of the current action.

**Description:** When invoked from a `DRAMA` action handler, reutrns the sequence count. This is set to 0 on the first invocation of an action and increment on each reschedule.

**Call:**
DitsGetSeq

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetSeq(n), Dits manual.

**Support:** Tony Farrell, `AAO`

---

## C.27 DitsGetTaskDescr — Return the description of the specified task.

**Function:** Return the description of the specified task.

**Description:** Returns the descripton of the task as set by the DitsSetDetails() function.

**Call:**
DitsGetTaskDescr `<taskname>`

**Language:** Tcl

**See Also:** `DTCL` manual, DitsSetDetails(3), DitsSetDetails(n), DitsGetTaskType(n), DitsGetTaskDescr(3).

**Support:** Tony Farrell, `AAO`

---

## C.28 DitsGetTaskName — Fetch the Dits name of the task running the interpeter

**Function:** Fetch the Dits name of the task running the interpeter

**Description:** Returns the name this task registed as.

**Call:**
DitsGetTaskName

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetTaskName(n), DitsAppInit(3), Dtcl_Init(3), Dtcl(1), Dtk(1).

**Support:** Tony Farrell, `AAO`

---

### C.29 DitsGetTaskType — Return the type of the specified task.

**Function:** Return the type of the specified task.

**Description:** Returns the type of the task as set by the DitsSetDetails() function.

**Call:**
DitsGetTaskType `<taskname>`

**Language:** Tcl

**See Also:** `DTCL` manual, DitsSetDetails(3), DitsSetDetails(n), DitsGetTaskType(3), DitsGetTaskDescr(n).

**Support:** Tony Farrell, `AAO`

---

### C.30 DitsKill — Kill an action in the current task.

**Function:** Kill an action in the current task.

**Description:** Kill an action in the current task. See DitsKillByIndex and DitsKillByName for more details.

**Call:**
DitsKill options

**Parameters:** none

**Options:**

**-name name** Specifies the name of the action to kill

**-index index** Specifies the index of the action to kill

**-arg arg** Specifies an argument structure to send with the signal. This argument is `NOT` deleted automatically.

**-status status** Specifies the status code (as an integer) to be associated with the action completion message. Defaults to `DITS__ACTKILLED`.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsKillByName(3), DitsKillByIndex(3), DitsGetActIndex(n).

**Support:** Tony Farrell, `AAO`

---

## C.31 DitsLoad — Loads and runs a task.

**Function:** Loads and runs a task.

**Description:** This routine is used to request that `DITS` load and run a new task. The calling routine specifies the task to be run and the machine on which it is to be run. This routine initiates a transation which uses the Imp Master task on the target machine to load the task. This means that the the imp mas- ter task must be running on the target machine. The imp master task is normally started with the dits_netstart com- mand, see Dits documentation for details.

**Call:**
DitsLoad program [options]

**Parameters:**

**(>) program (string)** Specifies the task/program to be run.

**Options:**

**-node name** The node to load the program on.

**-argument string** Arguments to be passed to the loaded task

**-process name** The process name to be given to the program

**-decode string** Argument decode string.

**-bytes integer**

**-priority priority** Specifies the requested priority for the program.

**-split** Split arguments using the decode string

**-relprio** Priority is relative to parent.

**-absprio** Priority is absolute.

**-symbol** For `VMS` target, name is a symbol

**-prog** For `VMS` target, name is a program name

**-names** For `VMS` target, inherit logical names and symbols. For Unix target, inherit environment variables.

**-wait** Wait for the operation to complete, using DitsActionWait to block this action. In this case the name the task registered under is returned instead of the transaction id.

**-waittimeout timeout** If -wait is specified, a timeout to be applied to the wait. Floating point seconds.

**Return value:** Transaction id.

**Language:** Tcl

**See Also:** `DTCL` manual,DitsLoad(3), DitsGetPath(n), DitsPutAction(n).

**Prior requirements:** Can only be invoked by Tcl commands executing in the context of a `DRAMA` action. Otherwise the obey/kick/pget/pset/control/monitor/ TaskRunning commands should be used.

**Support:** Tony Farrell, AAO

---

## C.32 DitsLogMsg — Log a message using the DRAMA logging facility.

**Function:** Log a message using the DRAMA logging facility.

**Description:** If DRAMA has an enabled logging system then this command calls the logLogMsg routine enabled by that logging system. The intention is to allow libraries and user code to log.

If no logging system is enabled, then this call does nothing.

The most common DRAMA logging system is GitLogger(3). Normally the application initialises GitLogger(), which tells DRAMA to use GitLogger(3) to write log messages.

**Call:**

DitsLogMsg <level> [<level>...] prefix message

**Parameters:**

**(>) level (char)** The logging level to be assoicated with this message. If one of the specified levels is enabled then the message is written to the log file (as determined by the loging system. The suppored levels are

| | |
|---|---|
| -startup | Message is a task startup/initialise log message. |
| -inst | Message concerns running instrumentation. |
| -user1 | User defined level 1. |
| -user2 | User defined level 2. |
| -all | Always log this message to the file. |

**(>) prefix (char)** A prefix string for the message, indicating say the library concerned.

**(>) message (char)** The message to log.

**Language:** Tcl

**See Also:** DTCL manual, DitsLogMsg(3), GitLogger(3), DitsLogSysEnabled(n).

**Support:** Tony Farrell, AAO

---

### C.33 DitsLogSysEnabled — Indicate if a DRAMA logging system might be active.

**Function:** Indicate if a DRAMA logging system might be active.

**Description:** If DRAMA has an enabled logging system then this command returns 1, Otherwise it returns 0.

The most common DRAMA logging system is GitLogger(3). Normally the application initialises GitLogger(), which tells DRAMA to use GitLogger(3) to write log messages.

This command just inquires if there is a logging system - through it is not impossible that this could be confused by an errant logging system (which does not use a clientData item correctly).

**Call:**
DitsLogSysEnabled

**Language:** Tcl

**See Also:** DTCL manual, DitsLogMsg(n), DitsLogMsg(3), GitLogger(3).

**Support:** Tony Farrell, AAO

---

### C.34 DitsLosePath — Lose the path to the specified tasks.

**Function:** Lose the path to the specified tasks.

**Description:** This commands causes the underlying Path to the specified tasks to be lost. This will shutdown all communication with that task and ensure the path got again cleanly when next commincation with it is attempted.

**Call:**
DitsLosePath taskname [...taskname]

**Parameters:**

**(>) taskname (string)** The name of the task, which must already be known to the system.

**Language:** Tcl

**See Also:** DTCL manual, DitsPathGet(3), DitsLosePath(3)

**Support:** Tony Farrell, AAO

---

## C.35 DitsMessage — Send a DRAMA obey/kick/get/set/control/monitor message.

**Function:** Send a DRAMA obey/kick/get/set/control/monitor message.

**Description:** A message of the specified type and name is sent to the specified task. The type defaults to OBEY. The action invoking this command will be reschduled for responses to this messages. The id of the transaction is returned.

**Call:**
DitsMessage task name [options]

**Parameters:**

**(>) task (string)** The name of the task to send the message too. We must already have a path to this task.

**(>) name (string)** The name of the action to send to this task.

**Options:**

**-arg arg** An Sds Id of an argument structure to attach to this message.

**-type type** The type of the message to send. One of OBEY/KICK/ GET/SET/CONTROL/MONITOR.

**-sendcur** Indicates the Send Current values flag should be set, which will cause monitor mesages to immediatly send the current values of any parameters.

**-repmonloss** cause the reporting of monitor messages which are lost due to waiting for buffer empty notification messages to arrive.

**-priority** Send the message as a priority message.

**-wait** Wait for the operation to complete, using DitsActionWait to block this action. In this case there is no outstanding transaction id on return so the values returned by DitsGetArgument is returned.

**-waittimeout timeout** If -wait is specified, a timeout to be applied to the wait. Floating point seconds.

**-trigger command** If -wait is enabled, then this option specifies a Tcl command to be executed when trigger messages occur. The argument to this command will be the Sds id of the trigger message argument.

**Return value:** The transaction id of the transaction if started.

**Language:** Tcl

**See also:** DitsInitiateMessage(3)

**Prior requirements:** Can only be invoked by Tcl commands executing in the context of a DRAMA action. Otherwise the obey/kick/pget/pset/control/monitor commands should be used.

**See Also:** DTCL manual, DitsInitiateMessage(3), DitsPutAction(n), DitsGetPath(n).

**Support:** Tony Farrell, AAO

### C.36 DitsMsgOut — Output a message to the user.

**Function:** Output a message to the user.

**Description:** Invoke the Dits level MsgOut call to return a message to the user which invoked the caller. If the caller is running in a DRAMA action, then the message is sent to the user which originated the action.

**Call:**
DitsMsgOut message

**Parameters:**

**(>) message (string)** The string to return to the user.

**Language:** Tcl

**See Also:** DTCL manual, DitsMsgOut(3), MessageHandler(n), DitsPutArgument(n), DitsPutActStatus(n).

**Support:** Tony Farrell, AAO

---

### C.37 DitsPutActStatus — Allows an TCL action code to set the DRAMA status.

**Function:** Allows an TCL action code to set the DRAMA status.

**Description:** The argument is a DRAMA status code which will become the status assoicated with the action which invoked this command.

This status is used if the action completes without a TCL error. If the action completes with a TCL error then the code DTCL__TCLCMD will be the action completion status.

This command only exists during action invocation and will always exist in that case in both the DRAMA and global name spaces (assuing namespaces are supported in the TCL version).

**Call:**
DitsPutActStatus status

**Parameters:**

**(>) status (int)** The DRAMA status code.

**Language:** Tcl

**Support:** Tony Farrell, AAO

**See Also:** DTCL manual, Dits manual, DitsPutAction(n), DitsPutRequest(n), DitsPutArgument(n).

---

## C.38 DitsPutAction — Add a new action to this task.

**Function:** Add a new action to this task.

**Description:** Adds a new action to the current task. When the action is invoked, the specified Tcl command will be invoked. This Tcl command can use either the normal C level calls to implement the action or the various Tcl level calls.

To reschedule from Tcl level, use DitsPutRequest and use the Tcl commands DitsPutObeyHandler and DitsPutKickHandler to change the handler commands.

To reschedule from C level, use the normal DitsPutObeyHandler/ DitsPutKickHandler/ DitsPutRequest C routines or if you wish from a C routine to specify Tcl routines for the next invocation of an action use DtclPutTclObeyHandler/DtclPutTclKickHandler. The use of these two routines rely on you having created this action with either this command or the C level call DtclPutActionHandler.

This command should only be invoked at `UFACE` context.

**Call:**
DitsPutAction name -obey obeyhandler [options]

**Parameters:**

**(>) name (string)** The name of the name action. Should not exceed 20 characters. Options

**-obey obeyhandler** Specifies the Tcl command to be invoked to handle obey messages

**-kick kickhandler** Specifies the Tcl command to be invoked to handle kick messages.

**-spawnable** If specified, the action can be spawned (Multiple outstanding actions of the same name are possible)

**-spawncheck checkhandler** If an action is spawnable, the name of a Tcl command to be executed to check if it is ok to spawn at this time. Returns successfully if yes, returns the appropiate error if not.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPutRequest(n), DitsPutRequest(3), DitsPutObeyHandler(n), DitsPutObeyHandler(3), DitsPutKickHandler(n), DitsPutKickHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DtclPutTclObeyHandler(n), DtclPutTclKickHandler(3), DtclPutAction(3), DitsPutActions(3).

**Support:** Tony Farrell, `AAO`

### C.39 DitsPutArgument — Allows an TCL action code to set the DRAMA completion argument

**Function:** Allows an TCL action code to set the DRAMA completion argument

**Description:** This command sets the argument associated with the completion of a DRAMA action.

This argument is used if the action complete and does so without a TCL error.

This command only exists during action invocation and will always exist in that case in both the DRAMA and global name spaces (assuing namespaces are supported in the TCL version).

**Call:**
DitsPutArgument id flag

**Parameters:**

**(>) id (int)** The SDS id of the argument.

**(>) flag (string)** One of COPY, DELETE or NODELETE, READFREE, FREDID. See DitsPutArgument(3) for details.

**Language:** Tcl

**Support:** Tony Farrell, AAO

**See Also:** DTCL manual, Dits manual, DitsPutAction(n), DitsPutRequest(n), DitsPutArgument(n).

---

### C.40 DitsPutKickHandler — Sets a new handler routine for the next kick

**Function:** Sets a new handler routine for the next kick

**Description:** Called by an Action routine to set the Tcl to be invoked on the next kick of this action.

This Tcl command should only be invoked from actions which where set up using the Tcl command DitsPutAction or the C routine call DtclPutAction.

**Call:**
DitsPutKickHandler command

**Parameters:** None

**Language:** Tcl

**See Also:** DTCL manual, DitsPutRequest(n), DitsPutRequest(3), DitsPutObeyHandler(n), DitsPutObeyHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DitsPutKickHandler(3), DtclPutTclObeyHandler(n), DtclPutTclKickHandler(3), DitsPutAction(n), DtclPutAction(3).

**Support:** Tony Farrell, AAO

---

## C.41 DitsPutObeyHandler — Sets a new handler routine for the next reschedule.

**Function:** Sets a new handler routine for the next reschedule.

**Description:** Called by an Action routine to set the Tcl to be invoked on the next reschedule of this action.

This Tcl command should only be invoked from actions which where set up using the Tcl command DitsPutAction or the C routine call DtclPutAction.

**Call:**
DitsPutObeyHandler command

**Parameters:** None

**Language:** Tcl

**See Also:** DTCL manual, DitsPutRequest(n), DitsPutRequest(3), DitsPutObeyHandler(3), DitsPutKickHandler(n), DitsPutKickHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DtclPutTclObeyHandler(n), DtclPutTclKickHandler(3), DitsPutAction(n), DtclPutAction(3).

**Support:** Tony Farrell, AAO

---

## C.42 DitsPutRequest — Request to the fixed part for when the Action returns.

**Function:** Request to the fixed part for when the Action returns.

**Description:** Called by an Action routine to indicate to the fixed part what it should do when the action returns

To help catch coding errors, it is an error to call this function from a command which is not operating in DRAMA OBEY of KICK context.

**Call:**
DitsPutRequest request [delay]

**Parameters:**

**(>) request (enum)** Indicates what is to be done. One of

| | |
|---|---|
| END | The action is to complete immediately |
| STAGE | The action is to reschedule immediately |
| WAIT | The action is to reqchedule on expiry of a timer. |
| SLEEP | The action is to be put to sleep. It can be worken up by a kick message or by a DitsSignal call. |
| MESSAGE | The action will be rescheduled on reception of a message from a subsiday action |
| EXIT | The action will complete and the task should exit. |

**(>) delay (float)** For WAIT/SLEEP/MESSAGE options, specify a delay in seconds before the action should be reschduled.

**Language:** Tcl

**Support:** Tony Farrell, AAO

**See Also:** DTCL manual, Dits manual, DitsPutRequest(3), DitsPutObeyHandler(n), DitsPutObeyHandler(3), DitsPutKickHandler(n), DitsPutKickHandler(3), DitsPutArgument(n), DitsPutActStatus(n), DtclPutTclObeyHandler(n), DtclPutTclKickHandler(3), DitsPutAction(n), DtclPutAction(3), DitsGetContext(n).

---

## C.43 DitsReason — Return the reason for a Dits Entry

**Function:** Return the reason for a Dits Entry

**Description:** Assuming the current procedure is invoked in the context of a Dits entry, this routine will return the reason for that entry.

The returned value is one of OBEY/KICK/TRIGGER/ASTINT/LOAD/LOADFAILED/MESREJECTED/ COMPLETE/DIED/PATHFOUND/PATHFAILED/MESSAGE/ERROR/EXIT/NOTIFY/ BULK_DONE/BULK_TRANSFERED. corresponding to the possible values of the variable reason in the routine DitsGetReason.

**Call:**
DitsReason

**Parameters:** None

**Language:** Tcl

**See Also:** DTCL manual, DitsReason(3), DitsGetEntInfo(3), Dits Manual

**Support:** Tony Farrell, AAO

---

## C.44 DitsRegistrationHandler — Specify a handler for task Registration messages.

**Function:** Specify a handler for task Registration messages.

**Description:** This routine specifies the name of a Tcl procedure which will be invoked whenever a task registers.

Note that any handler already put using DitsPutRegistrationHandler will we invoked after the Dtcl handler has been run.

The task must have invoked DitsAppInit() with the `DITS_M_REGISTRAR` flag set for this operation to succeed. This is done by default for the standard Dtcl/Dtk applications.

**Call:**
DitsRegistrationHandler [command]

**Parameters:**

**(>) command (string)** A Tcl command to be invoked when a task Connects The command will be appended with an argument which is a string containing the the name of the task which has registered and the node on which it has registered. If not supplied, any current handler is removed.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPutRegistrationHandler(3).

**Support:** Tony Farrell, `AAO`

---

## C.45 DitsRequestNotify — Requests notification when a Task's buffer empties

**Function:** Requests notification when a Task's buffer empties

**Description:** This command is used to obtain notification when a target task's buffer has emptied.

**Call:**
DitsRequestNotify task [options]

**Parameters:**

**(>) task (string)** The name of the task.

**Options:**

**-wait** Wait for the operation to complete, using DitsActionWait to block this action. In this case there is no outstanding transaction id on return so null is returned.

**-waittimeout timeout** If -wait is specified, a timeout to be applied to the wait. Floating point seconds.

**Return value:** The transaction id of the transaction which was initiated.

**Language:** Tcl

**See also:** DitsRequestNotify(3)

**Prior requirements:** Can only be invoked by Tcl commands executing in the context of a `DRAMA` action. Otherwise the obey/kick/pget/pset/control/monitor/ TaskRunning commands should be used.

**Support:** Tony Farrell, `AAO`

---

## C.46 DitsScanTasks — Scans the known tasks and returns details on them.

**Function:** Scans the known tasks and returns details on them.

**Description:** This routine scans the tasks known to `DRAMA` (`IMP`) on the current machine, returning details of each task.

The return value is a list of lists. Each outer list refers to one task. Each inner lists contains the following details for each task

| | |
|---|---|
| taskname | Name of the task concerned |
| local | Logical, true if task is local. |
| translator | Logical, true if the task is a translator task (a specialised task used for communicating with non-IMP tasks. task type `=>` An integer value set using the call DitsSetDetails. task description `=>` A character string set using the call DitsSetDetails. |
| current | Logical, true if the task is the current task |
| translated | Logical, true if the task is accessed though a translator task. (Only known for local tasks) |

**Call:**
DitsScanTasks

**Parameters:** none

**Language:** Tcl

**See Also:** `DTCL` manual, DitsScanTasks(3), DitsSetDetails(n), dtcl(1), dtk(1), Dtcl_Init(3), DitsAppInit(3),

**Support:** Tony Farrell, `AAO`

---

## C.47 DitsSetDebug — Set the Dits debug flag.

**Function:** Set the Dits debug flag.

**Description:** Used to enable output of Debugging information by Dits.

Returns the original value as an integer.

**Call:**

DitsSetDebug `levelmode` [mode...]

**Parameters:**

**(>) level (integer)** zero to turn debugging off, otherwise turn it on. The on values represent a bit mask to which th debugging level is set. You or the various bits to set a desrired value. See mode for the bit values

**(>) mode (char)** Alternative strings to control logging. The specified modes (more then one allows) will be enabled and all other modes turned off.

| | |
|---|---|
| `BASIC` | Basic stuff (1) |
| `IMPSYS` | Output full details of `IMP` Sys message handling (2) |
| `IMPINIT` | Log details of calls which initiate `IMP` Ops (4) |
| `MON` | Log monitor handling details (8) |
| `ACT` | Log action handling details (16) |
| `DETAILS` | Log details of such things as message lengths (32) |
| `BULK` | Log bulk data operatiosn (64) |
| `LIBS` | Turn on logging in libraries (128) |
| `IMPEVENTS` | Turn on `IMP` event logging (256) |
| `RXLPEXIT` | Receive loop exit (4096) |
| `SDSCHK` | Check for `SDS` leaks (32768) |
| `OFF` | Turn off only. Must be first and only. |

**Language:** Tcl

**See Also:** `DTCL` manual, DitsSetDebug(3)

**Support:** Tony Farrell, `AAO`

---

## C.48 DitsSetDetails — Set the task details.

**Function:** Set the task details.

**Description:** Sets the task type and task description for the task.

**Call:**
DitsSetDetails type [descr]

**Parameters:**

**(>) type (integer)** A user-defined integer value to be assocaited with the task. The default value is 0.

**(>) descr (string)** A string to be assocaited with the task.

**Language:** Tcl

**See Also:** DTCL manual, DitsSetDetails(3), DitsScanTasks(n).

**Support:** Tony Farrell, AAO

---

## C.49 DitsSetFixFlags — Set the DITS flags used for communicating with the DRAMA fixed part.

**Function:** Set the DITS flags used for communicating with the DRAMA fixed part.

**Description:** These flags are extra ways in which a DRAMA event handler can communicate with the DRAMA fixed part.

Returns the original value of the flags as an integer.

**Call:**
DitsSetFixFlags `maskflag` [flag...]

**Parameters:**

**(>) mask (integer)** An integer representing a bit mask to which the flags required. You or the various bits to set a desrired value. See mode for the bit values

**(>) mode (char)** Alternative strings to set the the flags . The specified flags (more then one allows) will be enabled and all other modes turned off.

| | |
|---|---|
| NO_SDS_CHECK | Disable SDS checks that might have been enabled by the DITS logging flags. (1) |
| OFF | Turn all off only. Must be first and only. |

**Language:** Tcl

**See Also:** DTCL manual, DitsSetDebug(3)

**Support:** Tony Farrell, AAO

---

### C.50 DitsSignal — Signal an action in the current task to reenter.

**Function:** Signal an action in the current task to reenter.

**Description:** Signal an action in the current task such that it reschedules. See DitsSignalByName and DitsSignalByIndex for more details.

**Call:**
DitsSignal options

**Parameters:** none

**Options:**

**-name name** Specifies the name of the action to signal

**-index index** Specifies the index of the action to signal

**-arg arg** Specifies an argument structure to send with the signal. This argument is deleted automatically after the action is invoked.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsSignalByName(3), DitsSignalByIndex(3), DitsGetActIndex(n).

**Support:** Tony Farrell, `AAO`

---

### C.51 DitsStatus — Returns the status of the Dits message which causes this entry.

**Function:** Returns the status of the Dits message which causes this entry.

**Description:** Assuming the current procedure is invoked in the context of a Dits entry, this routine will return the value corresponding to the value of the variable reasonStat in the routine DitsGetReason.

The value is returned as an integer.

**Call:**
DitsStatus

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetEntStatus(3), DitsReason(n).

**Support:** Tony Farrell, `AAO`

---

### C.52 DitsStatusText — Returns the status of the Dits message which causes this entry.

**Function:** Returns the status of the Dits message which causes this entry.

**Description:** Assuming the current procedure is invoked in the context of a Dits entry, this routine will return the value corresponding to the value of the variable reasonStat in the routine DitsGetReason.

The value is returned as the translated error code.

**Call:**
DitsStatusText

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetEntStatus(3), DitsReason(n), DitsErrorText(n), PutFacility(n), Mess package.

**Support:** Tony Farrell, `AAO`

---

### C.53 DitsTask — Fetch the name of the Dits task which sent this message.

**Function:** Fetch the name of the Dits task which sent this message.

**Description:** Assuming the current procedure is invoked in the context of a Dits entry, this routine will return the name of the task which sent the message.

**Call:**
DitsTask

**Parameters:** None

**Language:** Tcl

**See Also:** `DTCL` manual, DitsGetEntPath(3), DitsTaskFromPath(n), DitsAppInit(3), DitsPathGet(3).

**Support:** Tony Farrell, `AAO`

---

### C.54 DitsTaskIsLocal — Returns 1 if the specified task is local, 0 otherwise.

**Function:** Returns 1 if the specified task is local, 0 otherwise.

**Description:** Returns 1 if the specified task is local, 0 otherwise. We must already have got a path to this task.

**Call:**
DitsTaskIsLocal taskname

**Parameters:**

**(>) taskname (string)** The name of the task, which must already be known to the system.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPathGet(3), DitsTaskIsLocal(3).

**Support:** Tony Farrell, `AAO`

---

### C.55 DitsTaskNode — Return the node that a given task is on.

**Function:** Return the node that a given task is on.

**Description:** If we have path to the specified task, return the node name of the node on which the task is running.

**Call:**
DitsTaskNode

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPathToNode(3)

**Support:** Tony Farrell, `AAO`

---

### C.56 DramaLoad — Load a program using the Drama system.

**Function:** Load a program using the Drama system.

**Description:** Loads and runs a task using the Drama load system. For full details you should see the description of the DitsLoad() routine, to which this Tcl command provides an interface.

Note, prior to Tk 4.1, this command was named “load”.

**Call:**

DramaLoad program [options]

**Parameters:**

**(>) program (string)** The name of the program to run. See DitsLoad() documtation for full details of this specification

**Options:**

**-wait** Wait until the load has completed before returning.

**-complete command** Execute "command" when the load completes. (Only one of -wait or -complete can be used).

**-waitload** Wait but only until the load completes and the task has registered. Will return on first off - task registering, task load failure or task exiting.

**-loaded command** Execute "command" when the load completes succesfully. The command will be appended with three arguments, the program name as supplied above, the node on which the program was loaded and the task name the program has registered as.

**-failure command** Execute "command" if the load fails. The command will be appended with five arguments, the program name as supplied above, the node on which the program load was attempted, the dits error code translation, the target node error code and the dits error code as an integer.

**-exit command** Execute "command" when the loaded program exits. The command will be appended with three arguments, the program name as supplied above, the node on which the program was loaded and the text of the target node error code which the program returned. This will be the string `OK` if the program completed successfully.

**-node name** The node on which the program should be loaded. If not supplied, use the current node.

**-argument string** Arguments to the loaded program

**-process name** The process name to be given the the program

**-decode string** Argument decode string.

**-bytes integer** The number of bytes to allocated for the program.

**-priority integer** The requested priority for the program

**-timeout integer** Load timeout value

**-split** Split arguments using decode string

**-relprio** Priority is relative to parent

**-absprio** Priority is absolute

**-symbol** For `VMS` target, name is a symbol

**-prog** For `VMS` target, name is a program name

**-names** For `VMS` target, inherit logical names and symbols. For Unix target, inherit environment.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsLoad(3), Dits manual, Imp manual.

**Support:** Tony Farrell, `AAO`

---

## C.57 DtclError — Command invoked to process background errors.

**Function:** Command invoked to process background errors.

**Description:** The DtclError command doesn't exist as a built-in part of Dtcl. Instead, individual applications or users can define a DtclError command (e.g. as a Tcl procedure) if they wish to handle background errors. A Dtcl background error is one that occurs in a command that occurs while executing a specified handler command to a Dtcl command. For example, the success and error handlers to the Dtcl obey command. For a non-background error, the error can simply be returned up through nested Tcl command evaluations until it reaches the top-level code in the application; then the application can report the error in whatever way it wishes. When a background error occurs, the unwinding ends in the Dtcl library and there is no obvious way for Dtcl to report the error.

When Dtcl detects a background error, it invokes the the DtclError command, passing it the error message as its only argument. Dtcl assumes that the application has implemented the DtclError command, and that the command will report the error in a way that makes sense for the application. Dtcl will ignore any result returned by the DtclError command. If another Tcl error occurs within the DtclError command then Dtcl reports the error itself using Ers.

This form of error handling is almost exactly the same as is done by tk (see tkerror).

The Tk version of Dtcl includes a default DtclError procedure that behaves in a similar way to the default version of tkerror. It posts a dialog box containing the error message and offers the user a chance to see a stack trace that shows where the error occurred.

In a call to DtclError, the stack is found in the global variable errorInfo.

**Call:**

DtclError message

**Parameters:**

**(>) message (string)** The error message

**Language:** Tcl

**See Also:** `DTCL` manual, DtclBackgroundError(3), tkerror(n).

**Support:** Tony Farrell, AAO

---

## C.58 DtclFindFile — Find a file using search paths, defaultings and wildcards.

**Function:** Find a file using search paths, defaultings and wildcards.

**Description:** This command will return the a list names which can be used to open all files which meet the given file specification, after the application of defaults, wildcards and search paths. For example,

```
DtclFindFile ''FRED_DIR:*.c''
DtclFindFile FRED_DIR: -default ''*.c''
```

will both search the envionment variable/logical name search path FRED_DIR for all .c files and return names that be be used in an open statement. Under VMS, a search path is a logical name directory search path. Under UNIX/VxWorks, a search path is a colon separated list of directory names (in the format used by the PATH environment variable)

For application of defaults, each filename is broken up into three components, the directory spec, the filename and the file type. If a component is not supplied by the file specification, then it is taken from the default specification, if supplied.

Wildcards appropiate to the machine on which the program is being run can be specified.

Note the requirement for quotes around anything which may be interpeted by the Tcl.

**Call:**
DtclFindFile name [options]

**Parameters:**

**(>) name (string)** Name of file to find.

**Options:**

**-oneonly** Return the first file only.

**-default (string)** The string specifies the default file name.

**-nowild** Disallow wildcards.

**Language:** Tcl

**See Also:** DTCL manual, DulFindFile(3).

**Support:** Tony Farrell, AAO

---

## C.59 DtclParseFile — Parse a file name using defaults and search paths.

**Function:** Parse a file name using defaults and search paths.

**Description:** This program will return the a name which can be used to create the given file, after the application of defaults and search paths. For example,

```
dparsefile ''FRED_DIR:jack.c''
dparsefile -default ''*.c'' FRED_DIR:jack
```

will both return a name which will put the file jack.c into the first part of the search path defined by the `FRED_DIR` search path.

Under `VMS`, a search path is a logical name directory search path. Under `UNIX`/VxWorks, a search path is a colon separated list of directory names (in the format used by the `PATH` environment variable)

For application of defaults, each filename is broken up into three components, the directory spec, the filename and the file type. If a component is not supplied by the file specification, then it is taken from the default specification, if supplied.

Note the requirement for quotes around anything which may be interpeted by Tcl.

**Call:**
DtclParseFile name [options]

**Parameters:**

**(>) name (string)** Name of file to find.

**Options:**

**-default (string)** The string specifies the default file name.

**Language:** Tcl

**See Also:** `DTCL` manual, DulParseFile(3).

**Support:** Tony Farrell, `AAO`

---

## C.60 DtclTkDialog — Creates a dialog box containing a bitmap, message and one or more buttons

**Function:** Creates a dialog box containing a bitmap, message and one or more buttons

**Description:** This procedure is similar to tk_dialog, but is optimized for use in Dtcl/Dtk programs. In particular, is does not use tk_wait, which can cause problems. Instead, you must supply a command which will be invoked when a button is pressed.

Window positioning is handled by DtclTkDialogPosition - please see that command for details.

**Call:**

DtclTkDialog w title text [bitmap] default command arg1 [args...]

**Parameters:**

**(>) w (tk window name)** Window to use for dialog top-level. If the window name is prefixed with `!`, then the format is !class.w where class is the class name to be given to the dialog and .w is the window name.

**(>) title (string)** Title to display in dialog's decorative frame.

**(>) text (string)** Message to display in dialog.

**(>) bitmap (tk bitmap name)** Bitmap to display in dialog. An empty string means none.

**(>) default (integer)** Index of button that is to display the default ring. Use -1 to indicate no default.

**(>) command (string)** Command to be invoked when a button is pressed. It will have button number appended.

**(>) args (strings)** One or more strings to display in buttons across the bottom of the dialog box. There are three flag characters which can be pre-appended to the button string to change the behaviour. By default, the dialog is destroyed when the button is pressed. If button name is pre-appended with a dash (-), then the dash is removed and the dialog is not destroyed when the button is pressed. The three other flag ("`!`", "`*`", "+") are used to select pre-defined options for changine the display of the button. "`!`" indicates a Cancel button and uses appropiate colors. "`*`" Indicates an Ok button and uses appropiate colors. "+" indicates a special button and uses appropiate options (currently none). These colors can all be changed using the DtclTkDialogButOpts command.

**See Also:** `DTCL` manual, tk_dialog(n), DtclTkDialogButOpts(n)

**Support:** Tony Farrell, `AAO`

---

## C.61 DtclTkDialogButOpts — Get/Set the options for the predefined button types used by DtclTkDialog

**Function:** Get/Set the options for the predefined button types used by DtclTkDialog

**Description:** DtclTkDialog supports three predefined button types. These types used predefind options. The three types are "Cancel", "Ok" and "Special" buttons - and DtclTkDialog uses flag characters to specify that the appropiate pre-defined options are used.

This command allows you to Get or Set the options for the button type.

These settings take effect from the next call to DtclTkDialogButOpts.

By default, Cancel buttons have a white foreground can red background, whilst Ok buttons have a green background and white foreground. The special button type has no particular

defaults. Note, whilst colors are commonly changed, any options to the Tk "button" command can be specified

**Call:**
DtclTkDialogButOpts `CancelOkSpecial` [options...]

**Parameters:**

**(>) type (`CancelOkSpecial`)** The type of button for which we are setting or getting the options.

**(>) args (strings)** If supplied, the options to be used for that button.

**Example:**

```
DtclTkDialogButOpts Special -fg gold -bg black
```

This example sets the special button type (introduced by the + tag) to use a forground of gold and a background of black.

**See Also:** `DTCL` manual, DtclTkDialogButOpts()

**Support:** Tony Farrell, `AAO`

---

## C.62 DtclTkDialogCentre — Set the initial position of a dialog to the centre of the

**Function:** Set the initial position of a dialog to the centre of the current screen.

**Description:** Given a created dialog, this command will set the position of the dialog such that it is centred on the screen on which it is on.

**Language:** `TCL`

**Call:**
DtclTkDialogCentre window

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(>) window (tk_window)** The name of the dialog top-level window.

**Support:** Tony Farrell, `AAO`

---

### C.63 DtclTkDialogPosUnderMouse — Set the initial position of a dialog to be under the mouse

**Function:** Set the initial position of a dialog to be under the mouse

**Description:** Given a created dialog, this command will set the position of the dialog such that it is centred on current mouse position.

**Language:** `TCL`

**Call:**
DtclTkDialogPosUnderMouse window

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(>) window (tk_window)** The name of the dialog top-level window.

**Support:** Tony Farrell, `AAO`

---

### C.64 DtclTkDialogPosition — Set the initial position of a dialog.

**Function:** Set the initial position of a dialog.

**Description:** Given a created dialog, this command will set the position of the dialog such that it is centred on the screen or positioned under the mouse.

The default positioning is centered - except that if it appears you are running on a dual display system, it is centered on the display where the mouse currently is. (Some quesswork is involved here). If the positionioning mode is set to "under mouse" then that is where the dialog is positioned.

If more then $DtclTkDialogMaxWins (30) dialogs have appeared with no more then 2 second separations, then the dialog will `NOT` be displayed - a warning dialog is generated and the window name written to stderr.

Use the DtclTkDialogPositionSetUnderMouse or DtclTkDialogPositionSetCenter commands to change the positionioning mode.

**Language:** `TCL`

**Call:**
DtclTkDialogPosition window

**Parameters:** ("`>`" input, "`!`" modified, "W" workspace, "`<`" output)

**(>) window (tk_window)** The name of the dialog top-level window.

**Support:** Tony Farrell, `AAO`

---

## C.65 DtclTkHelpDialog — Creates a dialog box to display a simple help message.

**Function:** Creates a dialog box to display a simple help message.

**Description:** This command creates a simple display which displays some text. It is intended to support putting up simple help dialogs.

**Call:**
DtclTkHelpDialog w title text

**Parameters:**

**(>) w (tk window name)** Window to use for dialog top-level.

**(>) title (string)** Title to display in dialog's decorative frame.

**(>) text (string)** Message to display in dialog.

**See Also:** DtclTkDialog(n), tk_dialog(n).

**Support:** Tony Farrell, `AAO`

---

## C.66 DtclTkScreenInfo — Returns information about screens in a multi-desktop configuration.

**Function:** Returns information about screens in a multi-desktop configuration.

**Description:** The `X11` XInerama extension allows multiple screened to be considered as one. This generally works well, but sometimes we need to place windows sensibly in just one of these displays. Currently (2007, version

8. 4.14) Tcl/Tk provides no way of doing this through it may be added to the "winfo" command at some point. This command provides an interface to XInerama. If XInerama is not supported, then we consider that we just have one display.

If the window argument is not supplied, then returns 0/1 indicating if the XInerama is supported on this `OS`.

The window argument is used to find the `X11` display you are interested in. If supplied with no screen number, then returns the number of screens as known by Xinerama. If supplied with a screen number, returns a list being the x origin, y origin, x size and y size of the specified screen.

**Call:**
DtclTkScreenInfo [window [num]]

**Parameters:**

**(>) window (string)** The name of a window on the display in question.

**(>) num (integer)** A Xinerima screen number.

**Options:**

**Language:** Tcl

**See Also:** `DTCL` manual. Tcl/Tk winfo command, Tcl/Tk wm command. `X11` XInerama extension documentation.

**Support:** Tony Farrell, `AAO`

---

## C.67 ErrorHandler — Specify an error handler for Ers error messages

**Function:** Specify an error handler for Ers error messages

**Description:** This routine specifies the name of a Tcl procedure which will be invoked whenever an Ers error message is reported.

**Call:**
ErrorHandler [command]

**Parameters:**

**(>) command (string)** A Tcl command to be invoked when an error is reported. The command will be appended with an argument which is a string containing the error message (if there are more than one error messages they will be separated by newline characters in the string). The flags -highlight, -alarm and -bell may be appended if any of the reported messages had those flags set. If not supplied, the handler is reset to output messages to stderr.

**Language:** Tcl

**See Also:** `DTCL` manual, MessageHandler, Ers manual.

**Support:** Jeremy Bailey, `AAO`

---

## C.68 ErsAnnul — Annul any error messages reported with Ers.

**Function:** Annul any error messages reported with Ers.

**Description:** The various Dtcl commands may use ErsRep() to stack error messages. When used in the normal way, this is no problem, but if you wish to catch error returned by Dtcl commands, you will need explicity control over whether to report these messages.

ErsAnnul will annul all stacked messages so that they are not reported

**Call:**
ErsAnnul

**Language:** Tcl

**See Also:** DTCL manual, ErsAnnul(3) ErsFlush(n), ErsRep(n), DitsMsgOut(n), Ers manual

**Support:** Tony Farrell, AAO

---

### C.69 ErsFlush — Flush any error messages reported with Ers.

**Function:** Flush any error messages reported with Ers.

**Description:** The various Dtcl commands may use ErsRep() to stack error messages. When used in the normal way, this is no problem, but if you wish to catch error returned by Dtcl commands, you will need explicity control over whether to report these messages.

ErsFlush will cause all stacked messages to be reported to the user.

**Call:**
ErsFlush

**Language:** Tcl

**See Also:** DTCL manual, ErsFlush(3) ErsAnnul(n), ErsRep(n), DitsMsgOut(n), Ers manual

**Support:** Tony Farrell, AAO

---

### C.70 ErsRep — Report error messages.

**Function:** Report error messages.

**Description:** Report error messages using ErsRep. This messages are output when an action reschedules or when a UFACE routine returns.

**Call:**
ErsRep message [options]

**Parameters:**

**(>) message (string)** The message to report

**Options:**

**-bell** Suggest to the user interface that the Bell be rung when this message is output

**-highlight** Suggest to the user interface that this message be highlight when it is output.

**-alarm** Indicate to the user interface that this message is very important and should probably be acknowledged by the user.

**Language:** Tcl

**See Also:** `DTCL` manual, ErsRep(3) ErsAnnul(n), ErsFlush(n), DitsMsgOut(n), Ers manual

**Support:** Tony Farrell, `AAO`

---

## C.71 ImpProbe — Probe the Imp message layer of dits.

**Function:** Probe the Imp message layer of dits.

**Description:**

**Call:**
ImpProbe [options]

**Options:**

**-output cmd** Specify the Tcl command to be invoked to output each string line. The single argument is a line of text ImpProbe wants to output. If not supplied, The Dits routine MsgOut is used.

**-t task** As per ImpSysProbe -t option. The name a task has registered under. Full details for this task are output.

**-t pid** As per ImpSysProbe -t option. pid is an integer indicating the process id of the task concerned. Full details for this task are output.

**-b name**

**-b pid** Where name or pid are used to identify an imp task. In this case the only details output are those releating to the connections that tasks have with the specified task (including buffers used to communicate). This options may be specified multiple times.

**-nbd** Causes ImpSysProbe to list the context of the user noticeboard.

**Return value:** None

**Language:** Tcl

**See Also:** `DTCL` manual, ImpSysProbe(3), impdump(1), DitsScanTasks(n), impprobe(1).

**Support:** Tony Farrell, `AAO`

---

### C.72 IsUnix — Indicate if we are running on a Unix machine

**Function:** Indicate if we are running on a Unix machine

**Description:** Returns 1 if we the `TCL` program is running on a Unix machine, 0 otherwise.

**Call:**
IsUnix

**Language:** Tcl

**Support:** Tony Farrell, `AAO`

**See Also:** `DTCL` manual, IsVms(n), IsVxworks(n), Tcl variable tcl_platform (later versions of Tcl).

---

### C.73 IsVms — Indicate if we are running on a `VMS` machine

**Function:** Indicate if we are running on a `VMS` machine

**Description:** Returns 1 if we the `TCL` program is running on a `VMS` machine, 0 otherwise.

**Call:**
IsVms

**Language:** Tcl

**See Also:** `DTCL` manual, IsVms(n), IsVxworks(n), Tcl variable tcl_platform (later versions of Tcl).

**Support:** Tony Farrell, `AAO`

---

### C.74 IsVxworks — Indicate if we are running on a VxWorks machine

**Function:** Indicate if we are running on a VxWorks machine

**Description:** Returns 1 if we the `TCL` program is running on a VxWorks machine, 0 otherwise.

**Call:**
IsVxworks

**Language:** Tcl

**See Also:** `DTCL` manual, IsUnix(n), IsVms(n), Tcl variable tcl_platform (later versions of Tcl).

**Support:** Tony Farrell, `AAO`

---

## C.75 MessageHandler — Specify a handler for MsgOut messages

**Function:** Specify a handler for MsgOut messages

**Description:** This routine specifies the name of a Tcl procedure which will be invoked whenever an MsgOut message is reported.

**Call:**
MessageHandler command

**Parameters:**

**(>) command (string)** A Tcl command to be invoked when an MsgOut message is reported. The command will be appended with an argument which is a string containing the message. If not supplied, the handler reset to output messages to stdout.

**Language:** Tcl

**See Also:** `DTCL` manual, ErrorHandler(n), MsgOut(3), DitsMsgOut(n).

**Support:** Jeremy Bailey, `AAO`

---

## C.76 NotifyRequest — Requests notification when a Task's buffer empties

**Function:** Requests notification when a Task's buffer empties

**Description:** This command is used to obtain notification when a target task's buffer has emptied.

**Call:**
NotifyRequest name [options]

**Parameters:**

**(>) name (string)** The name of the task.

**Options:**

**-wait** Wait until the operation. has completed before returning. If -notify or -timeout handlers are supplied, then the result will be the result of the exeucted handler. If they are not supplied, then the result is a empty string for success and and error for a timeout.

**-notify command** Execute "command" when the notify message arrives. If not supplied and not waiting a message is output.

**-timeout integer** timeout value.

**-error command** Execute "command" if a timeout occurs. If not supplied and not waiting, a message is output.

**Language:** Tcl

**See Also:** DTCL manual, DitsRequestNotify(3), DitsRequestNotify(n).

**Support:** Tony Farrell, AAO

---

## C.77 PutFacility — A a new message code facility table to this program.

**Function:** A a new message code facility table to this program.

**Description:** The argument specifies the name of a file containing a message facility table (a _msgt.h) file. This facility is added to this program such that message codes in that facility can be translated.

**Call:**
PutFacility filename

**Parameters:**

**(>) filename (string)** Name of file containing the message facility table.

**Language:** Tcl

**See Also:** DTCL manual, DulReadFacility(3), Mess package.

**Support:** Tony Farrell, AAO

---

## C.78 SdpCreate — Create a parameter in the current task.

**Function:** Create a parameter in the current task.

**Description:** Create an item of the specified type and intial value in this tasks' parameter system.

**Call:**
SdpCreate name type value

**Parameters:**

**(>) name (string)** Name of the parameter to create. Ignored for SDS items.

**(>) type (string)** One of CHAR, BYTE, UBYTE, SHORT, USHORT, INT, UINT, FLOAT, DOUBLE, STRING, SDS. On 64 bit machines, INT64 and UINT64 are also supported.

**(>) value (varied)** Initial value of the parameter. Must match the type specified above.

**Language:** Tcl

**See Also:** DTCL manual, SdpCreate(3), SdpGet(n), SdpGetSds(n), SdpPut(n).

**Support:** Tony Farrell, AAO

---

## C.79 SdpGet — Get an item from the task's parameter system.

**Function:** Get an item from the task's parameter system.

**Description:** Read the value of a parameter in the task and return it.

In Tcl there is only a single version of SdpGet rather than a version for each type as in C — This is because in Tcl all variables are strings.

With the exception of string items, the length of the result is limited to TCL_RESULT_SIZE (200 bytes) and you will get an error if the result is longer (Say for arrays). You can use SdsArrayGet to access such items. For string items, the extra space is well defined and hence is handled here.

**Call:**

SdpGet name

**Parameters:**

**(>) name (string)** Name of parameter to get.

**Returns:** The value of the parameter item.

**Language:** Tcl

**See Also:** DTCL manual, SdpCreate(n), SdpGetString(3), SdpGetSds(n), SdpPut(n).

**Support:** Tony Farrell, AAO

**History:** 18-Apr-1997 - TJF - Original version 23-Feb-2015 - KS - Now uses Tcl_SetResult().

---

## C.80 SdpGetSds — Get an Sds reference to an item from the task's parameter system.

**Function:** Get an Sds reference to an item from the task's parameter system.

**Description:** Return an Sds id which refers to the named parameter. This allows you to access the parameter using normal Arg and Sds functions.

Note that a new Sds id is allocated and should be free-ed when you are finished with it by calling SdsFreeId. You should not delete this Sds item as doing so will delete the actual parameter.

**Call:**
SdpGet name

**Parameters:**

**(>) name (string)** Name of parameter to get.

**Returns:** An Sds id which refers to parameter item.

**Language:** Tcl

**See Also:** `DTCL` manual, SdpCreate(n), SdpGet(n), SdpGetSds(3), SdpPut(n), Sds manual.

**Support:** Tony Farrell, `AAO`

**History:** 18-Apr-1997 - `TJF` - Original version

---

## C.81 SdpPut — Put an item into the task's parameter system.

**Function:** Put an item into the task's parameter system.

**Description:** Put an item into the task's parameter system. We convert from a string to the type of the parameter concerned. If the conversion fails, we return an error.

**Call:**
SdpPut name value

**Parameters:**

**(>) name (string)** Name of parameter to get.

**(>) value (string)** New value for parameter. Must be convertable to the parameter type.

**Language:** Tcl

**See Also:** `DTCL` manual, SdpCreate(n), SdpGet(n), SdpGetSds(n), SdsPutString(3).

**Support:** Tony Farrell, `AAO`

**History:** 18-Apr-1997 - `TJF` - Original version

---

### C.82 SdpUpdate — Notify the parameter system that an item has been updated via its id.

**Function:** Notify the parameter system that an item has been updated via its id.

**Description:** The id is a value returned by SdpGetSds. It represents the Sds id of a parameter system item. This routine is used to notify the parameter system that the value of the parameter has been changed using the sds id.

**Call:**
SdpUpdate

**Parameters:**

**(>) id (SdsIdType)** A value returned by SdpGetSds

**(>) value (string)** New value for parameter. Must be convertable to the parameter type.

**Language:** Tcl

**See Also:** `DTCL` manual, SdpCreate(n), SdpGet(n), SdpGetSds(n), SdpUpodate(3).

**Support:** Tony Farrell, `AAO`

**History:** 12-Jul-2002 - `TJF` - Original version

---

### C.83 SdsArrayCell — Return the contents of one cell of an Sds array.

**Function:** Return the contents of one cell of an Sds array.

**Description:** The contents of the specified array cell are returned.

For scaler item arrays, the number of dimensions is not checked, except for there being no more then seven, since dimensions of scaler items don't really mean anything in Sds.

For structured arrays, the number of dimensions supplied must be at least the number in the structure. Excess dimensions are ignored. Note that for structures, the indexes start from 1.

For all scaler items, the contents of the specified cell are converted to a string. If the array is an array of structures, then the Sds id of the cell is returned.

**Call:**
SdsArrayCell id dim1 [dim2 dim3 dim4 dim5 dim6 dim7]

**Parameters:**

**(>) id (int)** The `SDS` identifier of the object

**(>) dimn (int)** The index to the cell.

**Language:** Tcl

**See Also:** DTCL manual, SdsArrayGet(n), SdsGet(n), SdsGet(3), Sds manual

**Support:** Tony Farrell, AAO

---

## C.84 SdsArrayGet — Return the contents of a 1 dimensional array

**Function:** Return the contents of a 1 dimensional array

**Description:** If the supplied Sds id describes a one dimensional array, return its contents converted to a string in the specified variable. If the object is an array of structures, then the ids' of the structures are returned.

By default, an array is setup in the variable who's name is specified. The element numbers are integers starting from 0.

The -list option will cause the result to be a list.

**Call:**

SdsArrayGet id name [options]

**Parameters:**

**(>) id (int)** SDS identifier of structure.

**(>) name (string)** The name of a variable in which the array contents will be returned.

**Options:**

**-list** The result variable should be setup as a list instead of an array.

**-string** As a special case, treat a 2 dimensional array of characters as an array of strings, where the first index is the maximum size of each string and the second is the number of items in the string array.

**-stringr** As per -string, but the dimension elements are reversed, the first being the number of elements in the array and the second the maximum size of each string.

**-element arrayelem** If name refers to a array variable and we want the result to be put in this element of this array. For -list is not specified, we construct new array element names of the form arrayelem,n where n is the index.

**Language:** Tcl

**See Also:** DTCL manual, SdsGet(n), SdsArrayCell(3), SdsGet(3), Sds manual

**Support:** Tony Farrell, AAO

---

### C.85 SdsCell — Find a component of a structured array.

**Function:** Find a component of a structured array.

**Description:** Given the indices to a component of a structure array, return an identifier to the component.

**Call:**
SdsCell array_id dims

**Parameters:**

() **array_id (>)** `SDS` id of the structured array

() **dims (>)** Indicates the array index where we are to insert the item. If more then one dimension, use a list and then the number of elements in the list must be the same as the number of array dimensions. An exception here is that you can treat all structure arrays as having only one dimension if desired (so that you could walk through all elements for example).

**Language:** Tcl

**See Also:** `DTCL` manual, Sds manual, SdsCell(3)

**Support:** Tony Farrell, `AAO`

---

### C.86 SdsCopy — Make a copy of an `SDS` object

**Function:** Make a copy of an `SDS` object

**Description:** Make a copy of an object and return an identifier to the copy. The copy is a new top level object, the original object is unchanged by the operation.

The object being copied can be either external or internal. The copy is always an internal object.

**Call:**
SdsCopy id

**Parameters:**

**(>) id (int)** The `SDS` identifier of the object to be copied.

**Returns:** The `SDS` identifier of the copy.

**Language:** Tcl

**See Also:** `DTCL` manual, SdsCopy(3), Sds manual

**Support:** Jeremy Bailey, `AAO`

---

### C.87 SdsDelete — Delete an SDS structure

**Function:** Delete an SDS structure

**Description:** Delete an SDS structure.

This routine does not free the id, normally you should consider using ArgDelete to do both jobs in one operation. This command is provided for consistency with C code.

**Call:**
SdsDelete id

**Parameters:**

**(>) id (int)** SDS identifier of structure to be deleted.

**Language:** Tcl

**Support:** Tony Farrelll, AAO

**See Also:** DTCL manual, Sds manual, ArgDelete(n), SdsFreeId(n), SdsFreeId(3), SdsDelete(3)

---

### C.88 SdsEvalAndCheck — Evaluate a Tcl command checking for SDS leaks.

**Function:** Evaluate a Tcl command checking for SDS leaks.

**Description:** The arguments to this command are combined and treated as a command to be exectued. The command is executed and a check is made for SDS ids leaking or being released during this command.

If the command completes successfully but SDS ids have leaked or been released, then an error is returned.

IF the command completes with an error and SDS ids have leaked or been released, then extra information is added to the error result.

Otherwise the return of this command is as per the argument command.

Note that in some cases it is NOT an error to leak or release SDS ids - if the command has a global impact in some way. This is up to the application design.

**Call:**
SdsEvalAndCheck <command...>

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, SdsNew(n), SdsGetExtra(n), SdsPutExtra(3).

**Support:** Tony Farrell, AAO

---

### C.89 SdsExtract — Extract an object from a structure.

**Function:** Extract an object from a structure.

**Description:**

Extract an object from a structure. The extracted object becomes a new independent top level object. The object is deleted from the original structure.

The identifier must not be that of a structure array component.

If the identifier is already that of a top level object, then the function does nothing

**Call:**
SdsExtract id

**Parameters:**

**(>) id (int)** The SDS identifier of the object to be extracted

**Language:** Tcl

**See Also:** DTCL manual, SdsExtract(3), Sds manual.

**Support:** Jeremy Bailey, AAO

---

### C.90 SdsFillArray — Fill out the contents of a structured array

**Function:** Fill out the contents of a structured array

**Description:** This routine will fill out an array of structures item with the copies of a specified struture.

**Call:**
SdsFillArray array_id id

**Parameters:**

**() array_id (>)** SDS id of the structured array into which we are to insert the items.

**() id (>)** The id of the item to fill the array with.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, SdsFillArray(3), SdsInsertCell(3).

**Support:** Tony Farrell, AAO

---

### C.91 SdsFind — Find an SDS structure component by name

**Function:** Find an SDS structure component by name

**Description:** Given the name of a component in an SDS structure return an identifier to the component.

**Call:**
SdsFind id name

**Parameters:**

**(>) id (int)** The SDS identifier of the structure.

**(>) name (string)** The name of the component.

**Returns:** The SDS identifier of the component.

**Language:** Tcl

**See Also:** DTCL manual, SdsFind(3), Sds manual.

**Support:** Jeremy Bailey, AAO

---

### C.92 SdsFindByPath — Find an Sds item by path name.

**Function:** Find an Sds item by path name.

**Description:** Given the name of a path to a component in an SDS structure return an identifier to the component.

**Call:**
SdsFindByPath id name

**Parameters:**

**(>) id (int)** The SDS identifier of the structure.

**(>) name (string)** The path name of the component.

**Returns:** The SDS identifier of the component.

**Language:** Tcl

**See Also:** DTCL manual, SdsFindByPath(3), Sds manual.

**Support:** Tony Farrell, AAO

---

### C.93 SdsFreeId — Free an identifier so that its associated memory may be reused

**Function:** Free an identifier so that its associated memory may be reused

**Description:** Each identifier allocated by SDS uses memory. To avoid excessive allocation of memory the SdsFreeId function can be used to free memory associated with an identifier that is no longer needed. When this is done the memory will be reused by SDS for a subsequent identifier when needed.

**Call:**
SdsFreeId id

**Parameters:**

**() id (int)** The SDS identifier to be freed.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgDelete(3), SdsFreeId(n), SdsFreeId(3), SdsDelete(n), ArgDelete(n)

**Support:** Jeremy Bailey, AAO

---

### C.94 SdsFreeIdCheck — Free an identifier so that its associated memory may be reused. Check id.

**Function:** Free an identifier so that its associated memory may be reused. Check id.

**Description:** Each identifier allocated by SDS uses memory. To avoid excessive allocation of memory the SdsFreeId function can be used to free memory associated with an identifier that is no longer needed. When this is done the memory will be reused by SDS for a subsequent identifier when needed.

This version will check that SdsDelete()/SdsFreeId() have been run, if required.

**Call:**
SdsFreeIdCheck id

**Parameters:**

**() id (int)** The SDS identifier to be freed.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgDelete(3), SdsFreeId(n), SdsFreeIdAndCheck(3), SdsDelete(n), ArgDelete(n)

**Support:** Tony Farrell, AAO

---

## C.95 SdsGet — Get data from an Sds item.

**Function:** Get data from an Sds item.

**Description:** Get data from scaler items.

The data is returned as an Tcl list, with all the items converted to string equivalents. If the type is `SDS_CHAR`, the values are converted to char representations.

The output of this routine makes no attempt to handle array dimensions, all data is considered to be one dimensional. You can use SdsInfo(n) to get array dimensions.

As a result of all the string conversions, this routine is not particularly efficient, especially for large arrays, You should consider if the job is better done by implementing new Tcl commands to handle your paricular case.

This routine cannot handle 64 bit intger types unless the machine on which it is running supports 64 bit long integer.

**Call:**

SdsGet id [options]

**Parameters:**

**() id (>)** `SDS` id of a primitive item to insert data into.

**Options:**

**-offset n** The offset in the `SDS` array to where we get the first item (item offset, not byte offset). Defaults to zero

**-max n** The maximum number of items to fetch. If not specified, we get all the values.

**Language:** Tcl

**See Also:** `DTCL` manual, Sds manual, SdsGet(3), SdsPut(n), ArgGet(n).

**Support:** Tony Farrell, `AAO`

---

## C.96 SdsGetExtra — Get an Sds structure's Extra infomation field contents.

**Function:** Get an Sds structure's Extra infomation field contents.

**Description:** Read bytes from the extra information field of an object. Note, this is treated as a null terminated string.

**Call:**

SdsGetExtra id [options]

**Parameters:**

**() id (>)** The id of the item.

**Options:**

**-len n** The maximum length to expect. If not supplied, we set this to the maximum length which can be returned by a normal Tcl command without special handling (`TCL_RESULT_SIZE`, normally 200 bytes)

**Language:** Tcl

**See Also:** `DTCL` manual, Sds manual, SdsNew(n), SdsPutExtra(n), SdsGetExtra(3).

**Support:** Tony Farrell, `AAO`

---

## C.97 SdsIndex — Find an `SDS` structure component by position

**Function:** Find an `SDS` structure component by position

**Description:** Given the index number of a component in an `SDS` structure return an identifier to the component.

**Call:**
SdsIndex id index

**Parameters:**

**(>) id (int)** The `SDS` identifier of the structure.

**(>) index (int)** Index number of the component to be returned Items in a structure are numbered in order of creation starting with one.

**Returns:** The `SDS` identifier of the component.

**Language:** Tcl

**See Also:** `DTCL` manual, SdsIndex(3), Sds manual

**Support:** Jeremy Bailey, `AAO`

---

## C.98 SdsIndexExists — Find if an `SDS` structure component specified by position exists.

**Function:** Find if an `SDS` structure component specified by position exists.

**Description:** Given the index number of a component in an `SDS` structure return true (1) if it exists and false (0) if it does not.

**Call:**
SdsIndexExists id index

**Parameters:**

**(>) id (int)** The SDS identifier of the structure.

**(>) index (int)** Index number of the component to be returned Items in a structure are numbered in order of creation starting with one.

**Returns:** The SDS identifier of the component.

**Language:** Tcl

**See Also:** DTCL manual, SdsIndex(3), Sds manual

**Support:** Tony Farrell, AAO

---

## C.99 SdsInfo — Return information about an SDS object.

**Function:** Return information about an SDS object.

**Description:** Given the identifier to an object, return the name, type, code and dimensions of the object.

Note, arguments from codevar onwards are optional.

**Call:**
SdsInfo id namevar codevar ndimsvar dimsvar

**Parameters:**

**(>) id (int)** The SDS identifier of the object.

**(>) namevar (string)** The name of the TCL variable to be set to the object name

**(>) codevar (string)** The name of the TCL variable to be set to the object type (optional).

**(>) ndimsvar (string)** The name of the TCL variable to be set to the number of dimensions (optional)

**(>) dimsvar (string)** The name of the TCL array variable to be set to dimensions. Indices 0, 1 .. 6 are used if required. If the item is not an array, then this variable is not set.

**Language:** Tcl

**See Also:** DTCL manual, SdsInfo(3), Sds manual.

**Support:** Tony Farrell, AAO

---

### C.100 SdsInsert — Insert an existing SDS object into a structure

**Function:** Insert an existing SDS object into a structure

**Description:** An existing top level object is inserted into a structure

**Call:**
SdsInsert parent_id id

**Parameters:**

**(>) parent_id (int)** The identifier of the structure to which the component is to be added.

**(>) id (int)** The SDS identifier of the object to be inserted.

**Language:** Tcl

**See Also:** DTCL manual, SdsInsert(3), Sds manual.

**Support:** Jeremy Bailey, AAO

---

### C.101 SdsInsertCell — Insert object into a structure array

**Function:** Insert object into a structure array

**Description:** Insert a top level object into a structure array at a position specified by its indices. Delete any object that is currently at that position.

**Call:**
SdsInsertCell array_id dims id

**Parameters:**

**() array_id (>)** SDS id of the structured array into which we are to insert the item.

**() dims (>)** Indicates the array index where we are to insert the item. If more then one dimension, use a list and then the number of elements in the list must be the same as the number of array dimensions. An exception here is that you can treat all structure arrays as having only one dimension if desired (so that you could walk through all elements for example).

**() id (>)** The id of the item to insert.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, SdsInsertCell(3), SdsInsert(n), SdsFillArray(n).

**Support:** Tony Farrell, AAO

---

### C.102 SdsList — List contents of an SDS object

**Function:** List contents of an SDS object

**Description:** A listing of the contents of an SDS object is generated on standard output. The listing consists of the name, type, dimensions and value of each object in the structure. The hierarchical structure is indicated by indenting the listing for components at each level.

For array objects only the values of the first few components are listed so that the listing for each object fits on a single line.

**Call:**
SdsList id [options]

**Parameters:**

**(>) id (int)** The SDS identifier of the object.

**Options:**

**-printcmd cmd** A command to be invoked to print each line of the listing. If specified, we call this command with the string appended for each line.

**-linelen n** When -printcmd is specified, this is the maximum number of characters to print.

**Language:** Tcl

**See Also:** DTCL manual, SdsList(3), ArgSdsList(3), Sds manual.

**Support:** Jeremy Bailey, AAO

---

### C.103 SdsNameExists — Determine if a named SDS structure component exists

**Function:** Determine if a named SDS structure component exists

**Description:** Given the name of a component in an SDS structure return true (1) if it exists and false (0) if it does not.

**Call:**
SdsNameExists id name

**Parameters:**

**(>) id (int)** The SDS identifier of the structure.

**(>) name (string)** The name of the component.

**Returns:** The SDS identifier of the component.

**Language:** Tcl

**See Also:** DTCL manual, SdsFind(3), Sds manual.

**Support:** Tony Farrell, AAO

---

### C.104 SdsNew — Create a new Sds structure

**Function:** Create a new Sds structure

**Description:** A new SDS structure is created it and an identifier to it is returned

**Call:**
SdsNew name [options]

**Parameters:**

**(>) name (string)** The name for the item.

**Options:**

**-type type** The Sds Type code. One of STRUCT, CHAR, BYTE, UBYTE, SHORT, USHORT, INT, UINT, I64, UI64, FLOAT and DOUBLE. Defaults to STRUCT.

**-parent parent_id** The id of a structured SDS Item which is to be the parent of this structure. If not supplied, then the new item will be a top level structure

**-extra info** Specifies extra information to be assocaited with the item.

**-dims dimensions** Specifies the dimensions of the item. "dimensions" is any array of up to seven positive integers which are the dimensions of the item if it is an array item. If not supplied, then the item is scaler.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, ArgNew(n), SdsNew(3).

**Support:** Tony Farrell, AAO

---

### C.105 SdsPhotoImages — A new Photo image format to support Tcl images read from Sds structures.

**Function:** A new Photo image format to support Tcl images read from Sds structures.

**Description:** Support has been added to the Tcl/Tk for a new Photo image format where the image can be stored in Sds structures. Both file and string match forms are supported. In the later case, the string is just the Sds id of an existing Sds structure. When a file is provided, it should contain an image.

Sds structures of type ImageStructure and any other Sds structure which looks like an image (the top level is a 2 dimensional array or a structure whose first item is a 2 dimensional array for mono-chrome. For colour, we will have a third dimension whose values is 3) is acceptable.

Currenlty, input data types must be `SDS_CHAR`, `SDS_BYTE`, `SDS_UBYTE`, `SDS_SHORT` or `SDS_USHORT`.

The name of the photo image format is "sds". When you write using this format, a Sds ImageStructure is created. It will be monochrome if possible, and will use a data type of `SDS_UBYTE`.

See Photo image format for details of usage.

**Language:** Tcl

**See Also:** `DTCL` manual, Sds manual, Dits specification, image(n), photo(n).

**Support:** Tony Farrell, `AAO`

---

### C.106 SdsPut — Put data into an Sds item.

**Function:** Put data into an Sds item.

**Description:** Write data into a primitive object. The data is supplied as a Tcl list.

Since Tcl works basically with string data, the data must be converted from strings to the desired type. Whilst this commands checks if each item is of the desired underlying type (char, integer or real) it does not check the range and overflows may occur when converting to the smaller types. In addtion, what happens if you attempt to convert negative values into unsigned values is undefined.

As a result of all the string conversions, this routine is not particuarly efficient, especially for large arrays, You should consider if the job is better done by implementing new Tcl commands to handle your paricular case.

This routine makes not attempt to handle array dimensions, all data is considered to be one dimensional.

This routine cannot handle 64 bit integer types unless the machine on which it is running supports 64 bit long integers.

**Call:**
SdsPut id data [options]

**Parameters:**

() **id (>)** SDS id of a primitive item to insert data into.

() **data (>)** The data to insert. Must be a single item or list of

() **items (for an array item)** of values of the correct type. For all SDS types, we first convert to "int". For floating point types, we first convert to "double". For CHAR types, should be a list of single characters (not a string).

**Options:**

**-offset n** The offset in the SDS array to where we put the first item. Defaults to zero

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, SdsPut(3), SdsGet(n), ArgPutString(n) ArgPutx(n)

**Support:** Tony Farrell, AAO

---

## C.107 SdsPutExtra — Put an Sds structure's Extra infomation field contents.

**Function:** Put an Sds structure's Extra infomation field contents.

**Description:** Read bytes from the extra information field of an object. Note, this is treated as a null terminated string.

**Call:**
SdsPutExtra id value

**Parameters:**

() **id (>)** The id of the item.

() **value (>)** The value to put (currently, only null terminated strings acceptable)

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, SdsNew(n), SdsGetExtra(n), SdsPutExtra(3).

**Support:** Tony Farrell, AAO

---

### C.108 SdsRead — Read an SDS object from a file

**Function:** Read an SDS object from a file

**Description:** An Sds object is read from a file and the identifier of the created external object is returned.

**Call:**
SdsRead file

**Parameters:**

**(>) file (string)** The name of the file to be read.

**Returns:** The SDS identifier of the resulting external object.

**Language:** Tcl

**See Also:** DTCL manual, SdsWrite(n), SdsReadFree(n), SdsRead(3), Sds manual.

**Support:** Jeremy Bailey, AAO

---

### C.109 SdsReadFree — Free Buffer allocated by SdsRead

**Function:** Free Buffer allocated by SdsRead

**Description:** SdsRead allocates a block of memory to hold the external object read in. This memory can be released when the object is no longer required by calling SdsReadFree (note that it is not possible to SdsDelete an external object).

If SdsReadFree is given an idientifier which was not produced by a call to SdsRead it will do nothing.

**Call:**
SdsReadFree id

**Parameters:**

**(>) id (int)** The SDS identifier of the object.

**Language:** Tcl

**See Also:** DTCL manual, SdsRead(n), SdsReadFree(3), Sds manual.

**Support:** Tony Farrell, AAO

---

### C.110 SdsRename — Rename an SDS object.

**Function:** Rename an SDS object.

**Description:** Specify a new name for an SDS object

**Call:**
SdsRename id name

**Parameters:**

**(>) id (int)** The SDS identifier of the object to be inserted.

**(>) name (string)** New name for object.

**Language:** Tcl

**See Also:** DTCL manual, SdsRename(3), Sds manual.

**Support:** Jeremy Bailey, AAO

---

### C.111 SdsResize — Change the dimensions of an array.

**Function:** Change the dimensions of an array.

**Description:** Change the number and/or size of the dimensions of an array. This operation can be performed on primitive arrays or structure arrays. Note that SdsResize does not move the data elements in the storage representing the array, so there is no guarantee that after resizing the array the same data will be found at the same array index positions as before resizing, though this will be the case for simple changes such as a change in the last dimension only.

If the size of a primitive array is increased the contents of the extra locations is undefined. Decreasing the size causes the data beyond the new limit to be lost.

If a structure array is extended the new elements created will be empty structures. If a structure array is decreased in size, the lost elements and all their components will be deleted.

**Call:**
SdsResize id dims

**Parameters:**

**() id (>)** SDS id of the structured array into which we are to insert the item.

**() dims (>)** Indicates the new dimensions. IF more then one dimension, use a list and then the number must be the same as the number of array dimensions required.

**Language:** Tcl

**See Also:** DTCL manual, Sds manual, SdsResize(3), SdsNew(n).

**Support:** Tony Farrell, AAO

---

## C.112 SdsWrite — Write an SDS object to a file

**Function:** Write an SDS object to a file

**Description:** Write an SDS object to a file - the object can be read back using SdsRead

**Call:**
SdsWrite id file

**Parameters:**

**(>) id (int)** The SDS identifier of the object.

**(>) file (string)** The name of the file to be written.

**Language:** Tcl

**See Also:** DTCL manual, SdsWrite(3), SdsRead(n), Sds manual.

**Support:** Jeremy Bailey, AAO

---

## C.113 TaskRunning — Checks if a task is running.

**Function:** Checks if a task is running.

**Description:** This command does a Dits GetPath operation to determine if the specified task can be accessed.

When this is determined (which may or may not be immediately) the results are written to the user using MsgOut.

**Call:**
TaskRunning name [options]

**Parameters:**

**(>) name (string)** The name of the task.

**Options:**

**-wait** Wait until the operation. has completed before returning. The command will in this case return 1 if the task is running and 0 otherwise. The MsgOut message will not be output.

**-running command** Execute "command" if it is determined if the task is running. The MsgOut message is now not output.

**-notrunning command** Execute "command" if it is determined if the task is not running. The command will be appended with two arguments, the status of operation as a text string and as an integer. The MsgOut message is now not output.

**-timeout integer** timeout value. If a timeout occurs, the a status of `DITS__APP_TIMEOUT` will be supplied to the notrunning command as its status arguments.

**Language:** Tcl

**See Also:** `DTCL` manual, DitsPathGet(3), DitsGetPath(n), obey(n), kick(n), pget(n), pset(n), control(n), monitor(n).

**Support:** Tony Farrell, `AAO`

---

## C.114 Translate — Perform one of a number of possible translations.

**Function:** Perform one of a number of possible translations.

**Description:** This command provides some general conversions required by astronomical software.

Note, some of these conversions use algorithms adapted from the starlink `SLALIB` library.

Where the returned value is a real number, the precision of the returned value is determined by the variable tcl_precision, which should be set to an integer value (See the Tcl/Tk book). If tcl_precision is not set (or is zero), then we use maximum number of significant digits available. (This is different from the general Tcl/Tk, itself, but is more sensible in this situation.)

**Call:**
Translate string options...

**Parameters:**

**(>) string (string)** The string to translate.

**Options:**

**-s2ra** Convert a string containing an Right Ascension in `HMS` to radians. Any single character non-numeric field separators are acceptable (other then a decimal point). The minutes and seconds fields may be omitted

**-ra2s** Convert an Right Ascension in radians to a string in `HMS`.

**-s2dec** Convert a string containing a Declination/Angle in `DMS` to radians. Any single character non-numeric field separators are acceptable (other then a decimal point). The minutes and seconds fields may be omitted

**-dec2s** Convert a Declination/Angle in radians to a string in DMS

**-sec2ra** Convert a Right Ascension in seconds of time (float) to radians.

**-sec2dec** Convert a Declination/Angle in seconds arc (float) to radians.

**-ndp n** For ra2s and dec2s options, specifies the number of decimal points on seconds. Defaults to 2.

**-sep char** For ra2s and dec2s options, specifies the separator. The default separator is ":".

**-sep2 char** For ra2s and dec2s, allows the second separator to different from the first. If must be specified after -sep, which will then be taken to specifiy the first separator.

**Language:** Tcl

**See Also:** DTCL manual, SlaLib, TranslateName(n).

**Support:** Tony Farrell, AAO

---

## C.115 TranslateName — Translate a logical name/environment variable.

**Function:** Translate a logical name/environment variable.

**Description:** Translate the value of the specified logical name/symbol (VMS) or environment variable (Unix/VxWorks). This command allows OS independent and efficient creation of file names etc.

**Call:**
TranslateName name [options]

**Parameters:**

**(>) name (string)** Name of the logical name or CLI symbol (VMS) or environement variable (Unix/VxWorks) to translate.

**Options:**

**-file (string)** We intend to access the specified file with name being the path to that file. Returns an appropriate file name for use in open calls

**-default (string)** If there is no such name, then use the specified string (file not attached) as the result. If default is not specified and the name does not exist, an error is returned.

**Language:** Tcl

**See Also:** DTCL manual, Translate(n), DtclFindFile(n), DtclParseFile(n).

**Support:** Tony Farrell, AAO

---

### C.116 after — Reimplementation of the alarm command under drama.

**Function:** Reimplementation of the alarm command under drama.

**Description:** Tk's after command could conflict with `DRAMA` timer requests. In addition, it is not available in `DTCL`, the command line version. Hence, alarm has been reimplemented to the same specifications.

**Call:**
after ms [args...]

**Parameters:**

**(>) ms (integer)** Number of milliseconds to trigger command after

**(>) args (strings)** Combined to get the command too execute. If not supplied, then the script pauses for the time specified. Note, you cannot pause scripts executing as a result of completion/success/error etc. handlers triggered by obey/kick/etc.

**Language:** Tcl

**Support:** Tony Farrell, `AAO`

---

### C.117 args — Create argument structure containing a number of strings

**Function:** Create argument structure containing a number of strings

**Description:** An `SDS` structure is created and components in the structure are created containing each of the arguments of the args command.

It returns the `SDS` identifier of the structure.

**Call:**
args arg1 arg2 .....

**Parameters:**

**(>) args (int)** Strings to be inserted into the `SDS` structure.

**Returns:** The `SDS` identifier of the created structure.

**Language:** Tcl

**See Also:** `DTCL` manual, ArgNew(n), ArgNew(3), ArgPutx(n), ArgDelete(n).

---

## C.118 control — Send a Control message to a Drama task

**Function:** Send a Control message to a Drama task

**Description:** A control message is sent to a Drama task. The command completes immediately the message is sent unless the -wait option is specified.

If any of the -success or -error options are not specified, default handlers will be invoked which cause messages to be output to the user.

**Call:**

control task type [args] [options]

**Parameters:**

**(>) task (string)** The name of the task in which the action is to be invoked. The form "task@node" may be used to send a message to a task on a remote machine.

**(>) type (string)** Type of control message to send

**(>) args (int)** Sds identifier of the argument structure to be supplied to the action (if required).

**Options:**

**-wait** Wait until the action has completed before returning. Recursive calls to obey/kick/etc. cannot be made in conjunction with -wait.

**-complete command** Execute "command" when the action completes. (Only one of -wait or -complete can be used).

**-success command** Execute "command" when the action completes succesfully. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the completion message. Note, the Sds structure referred to here is an external item which is deleted when your command returns. If you want to keep it about, you should copy it using SdsCopy.

**-error command** Execute "command" when the action completes with an error. The command will be appended with two arguments - the text of the status code returned from the task and the status code itself as an integer

**-node** The node on which the task resides. Assumes this is not specified in the task argument.

**-pathtimeout seconds** Specifies the timeout to apply to Get Path operations. The default is the -timeout value.

**-timeout seconds** Specifies the timeout to apply to actual messages. The default, -1, = no timeout. If a timeout is triggered, an error code of `DTCL__TIMEOUT` is supplied to the error handler.

**-deletearg** If an argument is supplied, the sds structure deleted and the id freeed when this routine is finished with it.

**-argfreeid** If an argument is supplied, the sds id is free-ed when this routine is finished with it.

**-argreadfree** If an argument is supplied, the sds id is read-free-ed and free-ed when the routine is finished with it.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**Language:** Tcl

**See Also:** `DTCL` manual, obey(n), kick(n), pget(n), pset(n), monitor(n), TaskRunning(n), DitsMessage(n), DuiExecuteCmd(3), DitsInitiateMessage(3).

**Support:** Jeremy Bailey, `AAO`

---

## C.119 kick — Send a kick message to an action in a Drama task

**Function:** Send a kick message to an action in a Drama task

**Description:** An kick message is sent to a Drama task. A kick message is used to cancel or otherwise modify the behaviour of an active action previously invoked by an obey message.

The command completes immediately the message is sent unless the -wait option is specified.

If the -success or -error options are not specified, default handlers will be invoked which cause messages to be output to the user.

**Call:**
kick task action [args] [options]

**Parameters:**

**(>) task (string)** The name of the task to which the message will be sent. The form "task@node" may be used to send a message to a task on a remote machine.

**(>) action (string)** Name of the action to be "kicked".

**(>) args (int)** Sds identifier of the argument structure to be supplied with the kick (if required).

**Options:**

**-wait** Wait until the kick has completed before returning. Recursive calls to obey/kick/etc. cannot be made in conjunction with -wait.

**-complete command** Execute "command" when the kick completes. (Only one of -wait or -complete can be used).

**-success command** Execute "command" when the kick completes succesfully. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the completion message. Note, the Sds structure referred to here is an external item which is deleted when your command returns. If you want to keep it about, you should copy it using SdsCopy.

**-error command** Execute "command" when the action completes with an error. The command will be appended with two arguments - the text of the status code returned from the task and the status code itself as an integer

**-node** The node on which the task resides. Assumes this is not specified in the task argument.

**-pathtimeout seconds** Specifies the timeout to apply to Get Path operations. The default is the -timeout value.

**-timeout seconds** Specifies the timeout to apply to actual messages. The default, -1, = no timeout. If a timeout is triggered, an error code of `DTCL__TIMEOUT` is supplied to the error handler.

**-deletearg** If an argument is supplied, the sds structure deleted and the id freeed when this routine is finished with it.

**-argfreeid** If an argument is supplied, the sds id is free-ed when this routine is finished with it.

**-argreadfree** If an argument is supplied, the sds id is read-free-ed and free-ed when the routine is finished with it.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**Language:** Tcl

**See Also:** `DTCL` manual, obey(n), pget(n), pset(n), control(n), monitor(n), TaskRunning(n), DitsMessage(n), DuiExecuteCmd(3), DitsInitiateMessage(3).

**Support:** Jeremy Bailey, `AAO`

---

## C.120 monitor — Send a monitor message to a Drama task

**Function:** Send a monitor message to a Drama task

**Description:** A monitor message is sent to a Drama task asking it to monitor parameters. The command completes immediately the message is sent unless the -wait option is specified.

If any of the -success, -error, -trigger or -info options are not specified, default handlers will be invoked which cause messages to be output to the user.

**Call:**

monitor task type args [options]

**Parameters:**

**(>) task (string)** The name of the task in which the monitor will to be invoked. The form "task@node" may be used to send a message to a task on a remote machine.

**(>) type (string)** The monitor message type - one of `START`, `FORWARD`, `ADD`, `DELETE`, `CANCEL`.

**(>) args (int)** Sds identifier of the argument structure to be supplied to the action (if required).

**Options:**

**-wait** Wait until the transaction has completed before returning. Recursive calls to obey/kick/etc. cannot be made in conjunction with -wait.

**-complete command** Execute "command" when the transaction completes. Only one of -wait or -complete can be used.

**-success command** Execute "command" when the transaction completes succesfully. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the completion message. Note, the Sds structure referred to here is an external item which is deleted when your command returns. If you want to keep it about, you should copy it using SdsCopy.

**-error command** Execute "command" when the action completes with an error. The command will be appended with two arguments - the text of the status code returned from the task and the status code itself as an integer

**-trigger command** Execute "command" when a trigger message is returned. The command will be appended with an argument which is the Sds identifier of an item containing the value of the modified parameter.

**-variable array** Specify the name of an array, elements of which are set as follows when monitor trigger messages arrive. Sets the array element `MONITOR_ID` to have the monitor transaction id. Sets array elements with names of each parameter to have the value of the corresponding parameter. Where the parameter is a one dimensional array of scalers or strings, the command SdsArrayGet is used to set the variable. The options -string and -element name, where name is the parameter name are specified. See that command for more details on the results. Where the parameter has more then one dimension (more then two for character array) or where it is a structured parameter, we try to convert the value to a string representation using the routine ArgToString and put this representation in the array element with the name of the parameter.

**-varcvtcmd command** This command is used to convert the value being put in the Tcl variable array element (-variable option above). For example, you may want to convert radians to hours:minutes:seconds. Standard style subsitutions are used to pass the name of the parameter (%n) and the value (%v) to this command. Use **%%** to pass a single percent sign. This command is only executed for non-array scaler items and character strings.

**-info command** Execute "command" when the action returns an informational or error message. The command will be appended with an argument which is the Sds identifier of the MsgStructure or ErsStructure returned.

**-node** The node on which the task resides. Assumes this is not specified in the task argument.

**-pathtimeout seconds** Specifies the timeout to apply to Get Path operations. The default is the -timeout value.

**-timeout seconds** Specifies the timeout to apply to actual messages. The default, -1, = no timeout. If a timeout is triggered, an error code of `DTCL__TIMEOUT` is supplied to the error handler.

**-deletearg** If an argument is supplied, the sds structure deleted and the id freeed when this routine is finished with it.

**-argfreeid** If an argument is supplied, the sds id is free-ed when this routine is finished with it.

**-argreadfree** If an argument is supplied, the sds id is read-free-ed and free-ed when the routine is finished with it.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**-sendcur** Send the current value of the parameter immediately.

**-repmonloss** cause the reporting of monitor messages which are lost due to waiting for buffer empty notification messages to arrive.

**Language:** Tcl

**See Also:** `DTCL` manual, obey(n), kick(n), pget(n), pset(n), control(n), TaskRunning(n), DitsMessage(n), DuiExecuteCmd(3), DitsInitiateMessage(3).

**Support:** Jeremy Bailey, `AAO`

---

## C.121 obey — Invoke an action in a Drama task

**Function:** Invoke an action in a Drama task

**Description:** An obey message is sent to a Drama task asking it to invoke an action. The command completes immediately the message is sent unless the -wait option is specified.

If any of the -success, -error, -trigger or -info options are not specified, default handlers will be invoked which cause messages to be output to the user.

**Call:**

obey task action [args] [options]

**Parameters:**

**(>) task (string)** The name of the task in which the action is to be invoked. The form "task@node" may be used to send a message to a task on a remote machine.

**(>) action (string)** Name of the action to be invoked.

**(>) args (int)** Sds identifier of the argument structure to be supplied to the action (if required).

**Options:**

**-wait** Wait until the action has completed before returning. Recursive calls to obey/kick/etc. cannot be made in conjunction with -wait.

**-complete command** Execute "command" when the action completes. Only one of -wait or -complete can be used.

**-success command** Execute "command" when the action completes succesfully. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the completion message. Note, the Sds structure referred to here is an external item which is deleted when your command returns. If you want to keep it about, you should copy it using SdsCopy.

**-error command** Execute "command" when the action completes with an error. The command will be appended with two arguments - the text of the status code returned from the task and the status code itself as an integer

**-trigger command** Execute "command" when the action returns a trigger message. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the trigger message.

**-info command** Execute "command" when the action returns an informational or error message. The command will be appended with an argument which is the Sds identifier of the MsgStructure or ErsStructure returned.

**-node** The node on which the task resides. Assumes this is not specified in the task argument.

**-pathtimeout seconds** Specifies the timeout to apply to Get Path operations. The default is the -timeout value.

**-timeout seconds** Specifies the timeout to apply to actual messages. The default, -1, = no timeout. If a timeout is triggered, an error code of `DTCL__TIMEOUT` is supplied to the error handler.

**-deletearg** If an argument is supplied, the sds structure deleted and the id free-ed when this routine is finished with it.

**-argfreeid** If an argument is supplied, the sds id is free-ed when this routine is finished with it.

**-argreadfree** If an argument is supplied, the sds id is read-free-ed and free-ed when the routine is finished with it.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**-kickarg var** If the action in the subsidary task is a spawnable action (meaning there may be multiple actions of the same name outstanding at any time) then if you want to kick that action, you need to supply a special first argument to the kick message (so that you can find the actual invocation you started). If this option is supplied, then

"var" is the name of a Tcl variable which will be set to the Sds id of an appropriate argument structure. If you specify this option, then you are responsible for doing an ArgDelete operation on the sds id. Note that this works only if you already have a path to the task in question. This can be ensured by using the TaskRunning command or any other message operation.

**Language:** Tcl

**See Also:** DTCL manual, kick(n), pget(n), pset(n), control(n), monitor(n), TaskRunning(n), DitsMessage(n), DuiExecuteCmd(3), DitsInitiateMessage(3).

**Support:** Jeremy Bailey, AAO

---

### C.122 pget — Get values of one or more parameters from a Drama task

**Function:** Get values of one or more parameters from a Drama task

**Description:** A parameter get or mget message is sent to a Drama task

The command completes immediately the message is sent unless the -wait option is specified.

If the -success or -error options are not specified, default handlers will be invoked which cause messages to be output to the user.

To access the value of the parameter a success handler must be used. Otherwise the value will simply be output to the user.

**Call:**

pget task param [param...] [options]

**Parameters:**

**(>) task (string)** The name of the task to which the message will be sent. The form "task@node" may be used to send a message to a task on a remote machine.

**(>) param (string)** Name of the parameters get. If one only parameter, then a get message is used, otherwise a mget message.

**Options:**

**-wait** Wait until the operation has completed before returning. Recursive calls to obey/kick/etc. cannot be made in conjunction with -wait.

**-complete command** Execute "command" when the operation completes. Only one of -wait or -complete can be used.

**-success command** Execute "command" when the operation completes succesfully. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the completion message. Note, the Sds structure referred to here is an external item which is deleted when your command returns. If you want to keep it about, you should copy it using SdsCopy.

**-error command** Execute "command" when the action completes with an error. The command will be appended with two arguments - the text of the status code returned from the task and the status code itself as an integer

**-node** The node on which the task resides. Assumes this is not specified in the task argument.

**-pathtimeout seconds** Specifies the timeout to apply to Get Path operations. The default is the -timeout value.

**-timeout seconds** Specifies the timeout to apply to actual messages. The default, -1, = no timeout. If a timeout is triggered, an error code of `DTCL__TIMEOUT` is supplied to the error handler.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**Language:** Tcl

**See Also:** `DTCL` manual, obey(n), kick(n), pset(n), control(n), monitor(n), TaskRunning(n), DitsMessage(n), DuiExecuteCmd(3), DitsInitiateMessage(3).

**Support:** Jeremy Bailey, `AAO`

---

### C.123 pset — Set a parameter in a task

**Function:** Set a parameter in a task

**Description:** A parameter set message is sent to a Drama task

The command completes immediately the message is sent unless the -wait option is specified.

If the -success or -error options are not specified, default handlers will be invoked which cause messages to be output to the user.

**Call:**

pset task param args [options]

**Parameters:**

**(>) task (string)** The name of the task to which the message will be sent. The form "task@node" may be used to send a message to a task on a remote machine.

**(>) param (string)** Name of the parameter to be set.

**(>) args (int)** Sds identifier of the argument structure containing the value to be set. The first component of the structure is the value used.

**Options:**

**-wait** Wait until the operation has completed before returning. Recursive calls to obey/kick/etc. cannot be made in conjunction with -wait.

**-complete command** Execute "command" when the operation completes. Only one of -wait or -complete can be used.

**-success command** Execute "command" when the operation completes succesfully. The command will be appended with an argument which is the Sds identifier of the argument structure returned in the completion message. Note, the Sds structure referred to here is an external item which is deleted when your command returns. If you want to keep it about, you should copy it using SdsCopy.

**-error command** Execute "command" when the action completes with an error. The command will be appended with two arguments - the text of the status code returned from the task and the status code itself as an integer

**-node** The node on which the task resides. Assumes this is not specified in the task argument.

**-pathtimeout seconds** Specifies the timeout to apply to Get Path operations. The default is the -timeout value.

**-timeout seconds** Specifies the timeout to apply to actual messages. The default, -1, = no timeout. If a timeout is triggered, an error code of `DTCL__TIMEOUT` is supplied to the error handler.

**-deletearg** If an argument is supplied, the sds structure deleted and the id freeed when this routine is finished with it.

**-argfreeid** If an argument is supplied, the sds id is free-ed when this routine is finished with it.

**-argreadfree** If an argument is supplied, the sds id is read-free-ed and free-ed when the routine is finished with it.

**-noflush** If an error is signalled immediately, then don't flush errors reported with Ers. This gives the caller (say catch) control over error output using ErsFlush and ErsAnnul. By default, errors will be flushed.

**Language:** Tcl

**See Also:** `DTCL` manual, obey(n), kick(n), pget(n), control(n), monitor(n), TaskRunning(n), DitsMessage(n), DuiExecuteCmd(3), DitsInitiateMessage(3).

**Support:** Jeremy Bailey, `AAO`