ANGLO-AUSTRALIAN OBSERVATORY AAO/DUL_SPEC_13
DRAMA Software Report 13
Version 0.5

Tony Farrell
23-Apr-97

DRAMA Utilities library

Contents

Revisions:

V0.0 07-Feb-1995 Original Version

V0.0.1 24-Feb-1995 Minor changes to command descriptions.

V0.5 23-Apr-1997 Minor update to documentation. Add “See Also” lines.

1 Introduction

The **DRAMA Utility Library** (**DUL**) is a collection of various utility programs and routines. It is meant to provide functions generally required of DRAMA programs but which are not required as part core of DRAMA. It is likely that some features of the **GIT** library will eventually be moved into **DUL**.

2 C++ support

If C++ is enabled when DRAMA is built, then the **DUL** library will include a package which supports the use of C++ with DRAMA programs. A separate document is provided for this package - see [1].

3 File Search Path Support

Various routines in this library provide support for file search paths and file name defaults. Some explaination of these techniques is required. In the `VMS` operating system, you can define an item called a "logical name" to point to a list of directories. You can then use this logical name in a file specification. This allows you to have a search path of directories in which to search for a given file.

Under `UNIX`, the shell's `PATH` "environment variable" performs basically the same function, but this feature is specific to the various command shells and any programs which required the feature had to reimplement it. Routines in this library allow the use of this feature for any "environment variable". In addition, they also support wildcarding and defaults. Wildcarding is not normally available in `UNIX` programs as is another feature normally implmented by the shell while defaulting needs a bit more explaination.

Assume you are writing a program which requires the user to specify an input file at some point. Now you may introduce a convention which says such files for this program normally end in "`.yak`". In addition, there may be both a default version of this file - "`defaults.yak`" and a local defaults version "`local_defaults.yak`" available in the directory "`/usr/local/yak`". So you now have a default directory - "`/usrlocal/yak`", a default name - "`defaults`" and a default type - "`.yak`". It would nice if, when prompting the user for their desired option, that that person does not have to specify parts for which defaults are known. For example, if the local defaults file - "`local_defaults.yak`" is required, the user should only have to enter "`local_defaults`".

The routines **DulFindFile()** and **DulParseFileName()** implement support for these features in a way that is portable accross all platforms supported by DRAMA.

A Detailed Subroutine Descriptions

The following are details of all the Dul routines which have been implemented at this stage.

A.1 DulErsAnnul — Annul the current stack of subsidary task messages.

Function: Annul the current stack of subsidary task messages.

Description: Annuls the current stack of subsidary task messages. After this call, DulErsRep() will have nothing to report until DulErsMessage() is called again to responed to a subsidary task `ERS` message.

Language: C

Call:
(Void) = DulErsAnnul (info,status);

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

(!) info (DulErsType *) Structure initialised by DulErsInit(3).

(!) status (StatusType *) Modified status. Cleared by this routine unless there is an error in the annul in which case it is set the that error.

Include files: Dul.h

See Also: Dul library document, Ers Document, DulErsInit(3), DulErsRep(3), DulErsFlush(3), ErsAnnul(3).

Support: Tony Farrell, `AAO`

A.2 DulErsFinished — Shutdown the DulErs system for this action.

Function: Shutdown the DulErs system for this action.

Description: This routine flushes and Ers messages held by the DulErs routines and reenables the default handling of subsidary task `ERS` messages, i.e., that they go directly to the user.

You need only invoke this routine if your action intends to continue rescheduling and is no longer interested in the Ers messages from subsidary actions.

Language: C

Call:
(Void) = DulErsFinished (info,status);

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

(!) info (DulErsType *) The DulInfo.

(!) status (StatusType *) Modified status.

Include files: Dul.h

See Also: Dul library document, Ers Document, DulErsInit(3), DitsInterested(3), DitsPutRequest(3), ErsRep(3), ErsFlush(3).

Support: Tony Farrell, `AAO`

A.3 DulErsInit — Initialise the DulErs system for this action.

Function: Initialise the DulErs system for this action.

Description: This routine should be invoked when ever an action wishes to manage Ers messages output by subsidary actions. The user should pass the address of an item of type DulErsType which is initialised by this routine and passed to other routines.

The action invoking this routine will now receive Ers messages from subsidary tasks (DitsGetEntReason(3) will return `DITS_REA_ERROR`). When it gets such messages, is should invoke DulErsMessage(3). Additionally, when the action completes, DulErsPutRequest(3) should be invoked to tidy up, instead of DitsPutRequest(3).

Language: C

Call:
(Void) = DulErsInit (info,status);

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

(<) info (DulErsType *)

(!) status (StatusType *) Modified status.

Include files: Dul.h

See Also: Dul library document, Ers Document, DulErsMessage(3), DulErsPutRequest(3), DulErsAnnul(3), DulErsRep(3), DitsGetEntReason(3), DitsInterested(3), DitsPutRequest(3), ErsRep(3), ErsAnnul(3).

Support: Tony Farrell, `AAO`

A.4 DulErsMessage — Handle subsidary task Ers messages.

Function: Handle subsidary task Ers messages.

Description: If DitsGetEntReason(3) returns `DITS_REA_ERROR`, then process the associated Ers message by storing it internally.

If you have invoked DulErsInit(), then normally you invoke this routine each time you get a message of type `DITS_REA_ERROR`, to add those messages to the stack of `ERS` messages

maintained by the info variable. You can later annul those messages using DulErsAnnul or report them using DulErsRep. This allow you to decide which of them is sent to the user rather then accepting the subsidiary task's output.

For example, you may decide that if the subsidary action completes with a specified error code, that condition can be handled by your task, so you stack any Ers messages reported by the task and then when you see the action completion message, you check the status. If the status was the one you are looking for, you can call DulErsAnnul() to annul of the messages. Otherwise, you call DulErsRep() or DulErsPutRequest() to cause the messages to be reported and output.

This facility provides the type of control of subsidiary task Ers messages that you have over your own Ers messages when using ErsPush(), ErsPop(), ErsAnnul() and ErsFlush().

Language: C

Call:

(Void) = DulErsMessage (info,status);

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

(`!`) info (DulErsType `*`) Structure initialised by DulErsInit(3).

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

Include files: Dul.h

See Also: Dul library document, Ers Document, DulErsInit(3), DitsGetEntReason(3), DulErsAnnul(3), DulErsRep(3), DulErsFlush(3), ErsRep(3), ErsAnnul(3).

Support: Tony Farrell, `AAO`

A.5 DulErsPutRequest — Tidy up DulErs stuff before calling DitsPutRequest(3).

Function: Tidy up DulErs stuff before calling DitsPutRequest(3).

Description: Having called DulErsInit(3), you should always use this routine instead of DulPutRequest(3) to put action completion codes. In addition, you should always put an action completion code, never rely on the automatic putting of the `DITS_REQ_END` code.

Note, this routine will do some work if status is bad on entry, being the tidying up of DulErs. It won't call DitsPutRequest() if status is bad. i

The reason for this routine is to ensure that DulErsRep' is invoked if the reason indicates the action is completing, i.e. `DITS_REA_END` and `DITS_REA_EXIT`.

Language: C

Call:

(Void) = DulErsPutRequest (info,request,status);

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

(!) info (DulErsType *) Structure initialised by DulErsInit(3).

(>) request (DitsReqType) The request, see DitsPutRequest(3).

(!) status (StatusType *) Modified status.

Include files: Dul.h

See Also: Dul library document, Ers Document, DulErsInit(3), DitsErsMessage(3), DulErsAnnul(3), DulErsRep(3).

Support: Tony Farrell, AAO

A.6 DulErsRep — Report the current stack of subsidary task messages using ErsRep(3).

Function: Report the current stack of subsidary task messages using ErsRep(3).

Description: This routine uses ErsRep(3) to report the current stack of subsidary task error messages. These messages will have been stacked using DulErsMessage(). Before and after calling this routine, you can use the standard Ers functions to control the messages.

Language: C

Call:

(Void) = DulErsRep (info,status);

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

(!) info (DulErsType *) Structure initialised by DulErsInit(3).

(!) status (StatusType *) Modified status. Note, the routine works when status is bad, as per ErsRep(3), but status may be set to a different status if the error reporting process fails.

Include files: Dul.h

See Also: Dul library document, Ers Document, DulErsInit(3), DulErsRep(3), DulErsFlush(3), ErsRep(3).

Support: Tony Farrell, AAO

A.7 DulFindFile — Find a file using wildcards and search paths.

Function: Find a file using wildcards and search paths.

Description: This routine is meant to find existing files using wildcarding, logical name/ environment variable search paths and defaults.

Two file specifications are supplied, the "filespec" to find and a "defaultspec". Each file specification is broken up into three components and components not supplied as part of "filespec" are taken from corresponding parts of "defaultspec".

File specifications are made up of one of the following formats

```
DIRECTORY_SPEC:name.type
DIRECTORY_SPEC/name.type
```

There are considered to be three components . The first component is the "`DIRECTORY_SPEC`", which specifies the directory in which to find the file. If neither the "filespec" or "defaultspec" specify a `DIRECTORY_SPEC`, then the current default directory is used.

The second component is "name". This consists of the string between the the last slash (or the colon) and the last period. If not supplied in either "filespec" or "defaultspec" then the wildcard "`*`" is supplied.

The third component is "type". This consists of the string between the the last period and the end of the string. If not supplied in either "filespec" or "defaultspec", then the wildcard "`*`" is supplied under `VMS` and nothing is supplied under Unix. Note that if this component is supplied under Unix/VxWorks in "defaultspec", then it is not possible to select a file which does not have a type. This restriction is required to get the defaulting effect normally required.

Since there may be multiple files which meet the specification, a context variable allows you to call this routine multiple times to return each file.

Both "defaultspec" and "filespec" are first processed through DulTranslate to translate environment variables or logical names represented using formats accepted by DulTranslate. Note that there is a limit of 1000 characters on the resulting translation. See DulTranslate for more details.

Unix/VxWorks Notes: If the directory specification ends in a colon then `DIRECTORY_SPEC` is considered to be an environment variable which will be translated. The result of this translation may be a colon separated list of directories which will be searched in order for the files. Alternately, for these operating systems, you may specify such a colon separated list directly.

Wildcards may be specified for the "name" and "type" components using standard `UNIX` shell style wildcards. The directory defaults to "./" - the current default.

`VMS` **Notes:** No longer supported (April 2018).

Language: C

Call:

(Void) = DulFindFile (fileSpec,defaultSpec,flags,resultSize, context,result,status);

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

(>) fileSpec (char *) The specification of the file to find. See above for details.

(>) defaultSpec (char *) The default specifications for the varoius components not supplied in in fileSpec. For components are not supplied here, the corresponding components in current_directory:`*.*` is used under `VMS` and current_directory:`*` under Unix/VxWorks. This argument is optional and my be specified as a null pointer.

(>) flags (int) A mask of flags. Possibilities are

`DUL_M_NOWILD`	Don't allow wildcards. Search paths will still be allowed.
`DUL_M_IMPDIR`	If a directory is found using the "`<name>:`" logical name translation format, then return in that format rather translated. This is useful if the name is to be used to load tasks via

() `IMP (DRAMA)` where the limit is 63 characters.

(>) resultSize (unsigned int) Size of the result string

(!) context (DulFindFileContextType **) If non-zero, enables multiple calls to find multiple matches. You should pass the address of a pointer to a variable of the specified type. On the first call you should set it to zero. If you use this argument, then you must call DulFindFileEnd when you are finished. If you pass the address of a non-zero pointer, it is assume this was set by a previous call to this routine and fileSpec and defaultSpec will be ignored.

(<) result (char *) The resultant file specification.

(!) status (StatusType *) Modified status. Will be `DUL__STRLEN` if the result string is too small. Will be `DUL__FILENOTFOUND` is no file is found. Will be `DUL__NOMORE` if there are no more files (subsequent searches). Other errors are possible from subsidiary routines.

Include files: DulFindFile.h

See Also: Dul library document, DulFindFileEnd(3), DulParseFileName(3), dfindfile(1).

Support: Tony Farrell, `AAO`

A.8 DulFindFileEnd — Finish finding files.

Function: Finish finding files.

Description: If you have been using DulFindFile to find multiple files, this routine releases the memory used for the context.

Language: C

Call:
(Void) = DulFindFileEnd (context,status);

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

(!) context (DulFindFileContextType *) The value of context returned by DulFindFileEnd. If this is zero, return immediately without error.

(!) status (StatusType *) Modified status.

Include files: DulFindFile.h

See Also: Dul library document, DulFindFile(3), DulParseFileName(3).

Support: Tony Farrell, `AAO`

A.9 DulFitsImgCreate — Define a new Photo image format to support Tcl images from Fits structures.

Function: Define a new Photo image format to support Tcl images from Fits structures.

Description: This function adds to Tcl/Tk, support for a new Photo image format where the image can read from Fits files.

The name of the photo image format is "fits". When you write using this format, a files file containing a monochrome image of unsigned short integer format is created.

Any fits file with a main data array which can be understood by the fitsio library can be read.

This image format should be enabled by invoking this routine. Note that it is only available if Dtcl has been built against Tk 4.0 or later and Dul has been built with `CFITSIO` support enabled. You will need to link your application against `CFITSIO` (`$CFITSIO_LIB`).

`WARNING:` As of 3-Jan-2017, no examples of this being used can be found in `AAT` Instrumentation software. As a result, this module is not being tested. `IF` used, please ensure you add testing.

Language: C

Call:
(void) = DulFitsImgCreate ()

Parameters: ("`>`" input, "`!`" modified, "W" workspace, "`<`" output) none

Include File: dul.h

See Also: Dul library document, DulFitsRead(3), photo(n), image(n), wish(n), SdsPhotoImages(n), dtk(n).

Support: Tony Farrell, `AAO`

A.10 DulFitsRead — Implements a Tcl Command to read a Fits file into an Sds structure.

Function: Implements a Tcl Command to read a Fits file into an Sds structure.

Description: This C function implements a Tcl command which will read a `FITS` file into an `SDS` structure. This function is only built if `CFITSIO` support is enabled when the `DRAMA DUL` Library is built. The Tcl command description follows

This command will read an image from a `FITS` file into an `SDS` structure. This structure will be of the form described in the `DITS` specification for ImageStructure type `SDS` structures.

If the "-type" option is not used, the file will be converted into an `SDS` structure of type `USHORT` (for historical compatiblity).

This command will only be available on Dtcl if `DRAMA` has been built with support for `CFITSIO` enabled. Dtcl/Dtk will include this command (from `DRAMA` version 1.5.2) if `CFITSIO` support is enabled when Dtcl is built.

The `FITS` header is also put into the `SDS` structure in an item named `HEADER`. This is a raw array of `SDS_CHAR` containing the raw `FITS` Header cards for the primiary header.

Tcl Command Call: `<user defined>` fitsFile [options]

Tcl Command Parameters: (`>`) fitsFile (string) The name of the `FITS` file to read.

Tcl Options:

-type type Specifies the `SDS` type into which the data should e converted. Possiblities are byte, short, ushort, int, uint, float, double, int64, natural. "natural" means that it is converted into the natural format for the file. If this argument is not supplied, then it is converted into ushort.

Tcl Command Returns: The `SDS ID` of the structure or `ERROR`.

Call:
DulFitsRead(clientData, interp, argc, argv)

Parameters: As per C implementation of Tcl commands.

Include Files: tcl.h, dul.h

Returns: Tcl `OK` or Error.

See Also: Dul library document, tclsh(n), DulFitsWrite(3), `FITSIO` library specification, `FITS` specification, Tcl/Tk book.

Language: C

Support: Tony Farrell, `AAO`

A.11 DulGetPathW — Get a path to a task, blocking the current action until it is complete.

Function: Get a path to a task, blocking the current action until it is complete.

Description: This function returns the path to the specified task, setting it up with the specified buffer sizes. If the action or `UFACE` context must wait for this to occur, then it is blocked (using DitsActionWait() or DitsUfaceWait). Such blocking allows the task to continue processing other actions and `UFACE` context messages.

In in an action context, 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 routine will return the status `DUL__UNEXPECTED` if this occurs.

Note that if another action/`UFACE` context calls this or assocaited routines (anything that invokes DitsActionWait/DitsUfaceWait) while this call is outstanding, then that call must unblock before this call will be unblocked - i.e. first in last out.

As of `DUL` version 3.40, we use the new DitsActionTransIdWait() and DitsUfaceTransIdWait() instead of the older versions which don't allow the transaction `ID` to be specified. This means we only return when the transaction completes or the timeout occurs, regardless of what other events come in. This is normally the required behaviour.

Language: C

Call:
(void) = DulGetPathW (task,node,flags,info,timeout,path,status)

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

- **(>) task (char *)** The name of the task to get a path to.
- **(>) node (char *)** The node the task is on. This is optional. If supplied as 0, then it is assumed that either the node is specified in the task argument (using stardard internet format) or the task is known locally.
- **(>) flags (int)** Flags to DitsPathGet. Se that routine for details.
- **(>) info (DitsPathInfoType *)** Use this structure to supply various details for the GetPath operation. See DitsPathGet for details

(>) timeout (double) If positive, this is a timeout to apply to the get path operation.

(<) path (DitsPathType *) The path is returned here.

(!) status (StatusType *) Modified status.

Include files: Dul.h

Prior requirements: Dits must have been initialised.

See Also: Dul library document, DulMessageW(3), DulLoadW(3), DitsPathGet(3), DitsActionTransIdWait(3), DitsUfaceTransIdWait(3), Dits specification.

Support: Tony Farrell, AAO

A.12 DulIntGetArgument — Gets the argument supplied when DulIntSignal() was invoked.

Function: Gets the argument supplied when DulIntSignal() was invoked.

Description: This call is used when an action handler which was signalled by a call to DulIntSignal(). It must be invoked at least once if DulIntSignal() was called with a non-zero value for the signal argument. It returns the signal value. You can call this routine multiple times to fetch the same value and no harm is done if you invoke it when the argument was zero.

This function returns the value obtained with DitsGetSigArg().

Language: C

Call:

(long int) = DulIntGetArgument ()

Include files: Dul.h

Prior Requirements: Should only be called from within a DRAMA action handler.

See Also: Dul library document, Dits specification, DulIntInit(3), DulIntSignal(3), DulIntTidy(3), DulIntGetSigArg(3), DitsSignalByIndexPtr(3).

Support: Tony Farrell, AAO

A.13 DulIntGetData — Get the client data item associated with a DulIntType item.

Function: Get the client data item associated with a DulIntType item.

Description: Just returns the clientData item associated with a DulIntType item when it was initialised.

Language: C

Call:
(Void *) = DulIntGetData (v)

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

(>) v (DulIntType) The variable describing the action to signal.

Include files: Dul.h

See Also: Dul library document, Dits specification, DulIntInit(3), DulIntSignal(3), DulIntTidy(3),

Support: Tony Farrell, AAO

A.14 DulIntISR — A complete ISR which invokes DulIntSignal().

Function: A complete ISR which invokes DulIntSignal().

Description: In some cases, the flexibily provided by DulIntSignal() is not required (you can pass an argument and check status). In those cases, this wrap up can be used. See DitsIntSignal() for full details of the behaviour of this routine.

Note that as DulIntType is a pointer type, you can cast this function to any function type which takes a pointer compatible single argument. This will often be required if using this routine directly as an ISR.

Language: C

Call:
(void) = DulIntISR (v)

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

(>) v (DulIntType) The variable describing the action to signal.

Include files: Dul.h

See Also: Dul library document, Dits specification, DulIntInit(3), DulIntSignal(3), DulIntTidy(3), DitsSignal(3).

Support: Tony Farrell, AAO

A.15 DulIntInit — Initialise a DulIntType variable for this action.

Function: Initialise a DulIntType variable for this action.

Description: A variable of type DulIntType is initialised to refer to this action. You can then pass this variable (which is a pointer type) to an interrupt service routine where you can use it with a call to DulIntSignal() to signal this action. (DulIntISR is a wrap up of DulIntSignal() proving a complete implementation of a simpile `ISR` which takes a single parameter).

This DulIntType variable can be used with multiple interrupt events, but you must call DulIntTidy() when you are finished with it otherwise you will have a memory leak. Normally, you would call DulIntTidy() before action completion although as long as the action which calls this routine is not spawnable, you can use the variable across invocations of the action.

In addition, the DulIntSignal() call can be invoked anywhere it is necessary to signal the action which invoked DulIntInit().

This function mallocs a structure. It stores in the structure the task id fetched by DitsGetTaskId(), the action index returned by DitsGetActIndex() and the clientData item supplied by the user.

Language: C

Call:
(Void) = DulIntInit (clientData,v,status)

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

(>) clientData (void *) An item to be associated with the DulIntType variable, it can be fetched using DulIntGetData().

(<) v (DulIntType *) The variable to be initialised.

(!) status (StatusType *) Modified status.

Include files: Dul.h

Prior Requirements: Should only be called from within a `DRAMA` action handler.

See Also: Dul library document, Dits specification, DulIntGetData(3), DulIntSignal(3), DulIntTidy(3), DulIntISR(3), DulIntGetArgument(3), DitsGetTaskId(3), DitsSignal(3), DitsGetActIndex(3).

Support: Tony Farrell, `AAO`

A.16 DulIntSignal — Signal the action which created the DulIntType variable.

Function: Signal the action which created the DulIntType variable.

Description: Signal the action which created the DulIntType variable. The action will rescheduled with with a reason of `DITS_REA_ASTINT`.

The integer argument becomes the action argument and can be fetched using DulIntGetArgument().

A wrap of this routine is provided by DulIntISR(). This wrap up could be used as a complete `ISR` whilst DulIntSignal(3) allows you to send an argument and check status.

This routine uses DitsEnableTask() to enable correct `DRAMA` task context, then uses DitsSignal() to signal that task and DitsRestoreTask() to restore the task context.

This routine is designed so that it can be invoked from VxWorks interrupt service routnes, but it can be used any time it is required to signal an action.

Normally, any errors are reported using ErsRep(), and status is set bad. But when run in a VxWorks interrupt service routine, the VxWorks routine will be used to report a message to the console. This is also done if this routine is used in a different VxWorks task from the task which is being signalled.

Language: C

Call:
(Void) = DulIntSignal (v,arg,status)

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

(>) v (DulIntType) The variable describing the action to signal.

(>) arg (long int) Argument to be passed to the action.

(!) status (StatusType *) Modified status.

Include files: Dul.h

See Also: Dul library document, Dits specification, DulIntInit(3), DulIntGetData(3), DulIntTidy(3), DulIntISR(3), DulIntGetArgument(3), DitsSignalByIndexPtr(3), DitsEnableTask(3), DitsRestoreTask(3).

Support: Tony Farrell, `AAO`

A.17 DulIntTidy — Tidy up a DulIntType structure.

Function: Tidy up a DulIntType structure.

Description: Tidys up a DulIntType variable by freeing the memory it is using. Note, after this call you can no longer user this variable.

Language: C

Call:
(Void) = DulIntTidy (v,status)

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

(>) v (DulIntType) The variable describing the action to signal.

(>) v (DulIntType) The variable to be initialised.

(!) status (StatusType *) Modified status.

Include files: Dul.h

See Also: Dul library document, Dits specification, DulIntInit(3), DulIntGetData(3), DulIntSignal(3), DulIntISR(3), DulIntGetArgument(3), DitsGetTaskId(3), DitsSignal(3), DitsGetActIndex(3).

Support: Tony Farrell, `AAO`

A.18 DulLoadFacs — Loads the standard facilites and user specified ones.

Function: Loads the standard facilites and user specified ones.

Description: The message facility table file is a C include file is generated by the messgen program and defines a couple of structures which are normally passed directly to MessPutFacility.

This function loads all the standard `DRAMA` facility tables if required and any the include files of which are specified in the environment variable `DRAMA_FACILITIES`.

The `DRAMA_FACILITIES` environment variable is a space seperated list of facility table include files. The names can include environment variables in the format `DIRECTORY`:file where `DIRECTORY` is an environment variable which translates to a directory. An example of the value of the variable might be

"`GCAM_DIR`:gcam_err_msgt.h `GCAM_DIR`:vfg_err_msgt.h"

Language: C

Call:
(void) = DulLoadFacs (status)

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

(>) status (StatusType *) Modified status

Include File: dul.h

External functions used:

See Also: Dul library document, DulReadFacility(3), MessPutFacility(3), Mess package spec.

Support: Tony Farrell, AAO

A.19 DulLoadW — Load a task and get a path to it, blocking until it is complete.

Function: Load a task and get a path to it, blocking until it is complete.

Description: This function returns the path to the specified task, setting it up with the specified buffer sizes. If the does not existing, then we will try to load it using the supplied information.

If the action or UFACE context must wait for this to occur, then it is blocked (using DitsActionWait() or DitsUfaceWait). Such blocking allows the task to continue processing other actions and UFACE context messages.

In in an action context, 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 routine will return the status DUL__UNEXPECTED if this occurs.

Note that if another action/UFACE context calls this or assocaited routines (anything that invokes DitsActionWait/DitsUfaceWait) while this call is outstanding, then that call must unblock before this call will be unblocked - i.e. first in last out.

As of DUL version 3.40, we use the new DitsActionTransIdWait() and DitsUfaceTransIdWait() instead of the older versions which don't allow the transaction ID to be specified. This means we only return when the transaction completes or the timeout occurs, regardless of what other events come in. This is normally the required behaviour.

Language: C

Call:

(void) = DulLoadW (taskname,node,PathFlags,info,program,ArgString, Flags,TaskParams,timeout,path,s

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

(>) taskname (char *) The name of the task to get a path to. We use this name to attempt to find the task before loading it, if we don't find it.

(>) node (char *) The node the task is on or is to be loaded on. This is optional. If supplied as 0, then it is assumed that either the node is specified in the taskname

() argument (using stardard internet format) or the task is known locally.

(>) PathFlags (int) Flags to DitsPAthGet. Se that routine for details.

(>) info (DitsPathInfoType *) Use this structure to supply various details for the GetPath operation. See DitsPathGet for details

(>) program (char *) Specifies the program to be run. See

() **DitsLoad ()** for a more detailed description of how a task may be specified.

(>) **ArgString (char *)** Arguments to be passed to the loaded task, formatted into a single character string. See DitsLoad() for a more detailed discussion of task arguments.

(>) **Flags (long int *)** A flag word controlling some options. In general these options are system-dependent, at least to some extent. See DitsLoad for more details.

(>) **TaskParams (DitsTaskParamType *)** Address of a structure containing additional values used to control the loading of the task. The use of the various fields in this structure is controlled by the setting of the bits in the Flags argument.

(>) **timeout (double)** If positive, this is a timeout to apply each operation (loading and getting path).

(<) **path (DitsPathType *)** The path is returned here.

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

Include files: Dul.h

Prior requirements: Dits must have been initialised.

See Also: Dul library document, DulGetPathW(3), DulMessageW(3), DitsLoad(3), DitsPathGet(3), DitsActionTransIdWait(3), DitsUfaceTransIdWait(3), Dits specification.

Support: Tony Farrell, `AAO`

A.20 DulMessageW — Send a message to task, blocking the current action until it is complete.

Function: Send a message to task, blocking the current action until it is complete.

Description: This function sends a message to a task. The invoking action or `UFACE` context is blocked (using DitsActionWait() or DitsUfaceWait). to await the completion of this message. Such blocking allows the task to continue processing other actions and `UFACE` context messages.

In in an action context, 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 routine will return the status `DUL__UNEXPECTED` if this occurs.

Note that if another action/`UFACE` context calls this or associated routines (anything that invokes DitsActionWait/DitsUfaceWait) while this call is outstanding, then that call must unblock before this call will be unblocked - i.e. first in last out.

As of `DUL` version 3.40, we use the new DitsActionTransIdWait() and DitsUfaceTransIdWait() instead of the older versions which don't allow the transaction `ID` to be specified. This means we only return when the transaction completes or the timeout occurs, regardless of what other events come in. This is normally the required behaviour.

MsgOut and ERS messages are forwarded (as of version 3.55).

Warning 1: When run in UFACE context when outside the DRAMA message receive loop (say DitsMainLoop() etc. has not been called) then DitsGetArgument() will allways be nil on return from this routine. In other cases (actions content and UFACE context in reponse to a message) on return from this routine, DitsGetArgument() have the return argument if any.

The inconsistency is due to a potential resource leak. Please see DulMessageWArg() to avoid this problem.

Warning 2: If the target action is expected to return a large argument structure, then this function may be quite inefficent as it does an SdsCopy on the structure. It is suggested that normal resheduling be used in such cases.

Language: C

Call:
(void) = DulMessageW (type,path,name,argument,timeout,status)

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

(>) type (DitsMsgType) The type of message to send. One of

DITS_MSG_KICK	Kick an action
DITS_MSG_OBEY	Start an action
DITS_MSG_GETPARAM	Get a parameter value
DITS_MSG_SETPARAM	Set a parameter value
DITS_MSG_CONTROL	Dits control message
DITS_MSG_MONITOR	Dits parameter monitor message

See DitsInitiateMessage for full details of each message type.

(>) path (DitsPathType *) The path to the task as returned by DitsGetPath or DulGetPathW.

(>) name (char *) The name associated with the message. Depends on message type but for OBEY and KICK, this is the action name.

(>) argument (SdsIdType) An argument to the message. This should be an Sds id. See DitsGetArgument and DitsPutArgument for more details on action arguments.

(>) timeout (double) If positive, this is a timeout to apply to the obey operation.

(!) status (StatusType *) Modified status.

Include files: Dul.h

Prior requirements: Dits must have been initialised.

See Also: Dul library document, DulGetPathW(3), DulLoadW(3), DitsInitiateMessage(3), DitsActionWait(3), DitsUfaceWait(3), DulMessageWArg(3), Dits specification.

Support: Tony Farrell, AAO

A.21 DulMessageWArg — Send a message to task, blocking the current action until it is complete.

Function: Send a message to task, blocking the current action until it is complete.

Description: This function sends a message to a task. The invoking action or UFACE context is blocked (using DitsActionWait() or DitsUfaceWait). to await the completion of this message. Such blocking allows the task to continue processing other actions and UFACE context messages.

In in an action context, 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 routine will return the status DUL__UNEXPECTED if this occurs.

Note that if another action/UFACE context calls this or associated routines (anything that invokes DitsActionWait/DitsUfaceWait) while this call is outstanding, then that call must unblock before this call will be unblocked - i.e. first in last out.

Unlike DulMessageW(), the return argument (if any) is available in the argOut parameter. The user must call SdsDelete() and SdsFreeId on this when finished with it. (This allows the argument to be accessed in cases where DulMessageW() can't get at it.

As of DUL version 3.40, we use the new DitsActionTransIdWait() and DitsUfaceTransIdWait() instead of the older versions which don't allow the transaction ID to be specified. This means we only return when the transaction completes or the timeout occurs, regardless of what other events come in. This is normally the required behavior.

MsgOut and ERS messages are forwarded (as of version 3.55).

Warning: If the target action is expected to return a large argument structure, then this function may be quite inefficient as it does an SdsCopy on the structure. It is suggested that normal rescheduling be used in such cases.

Language: C

Call:

(void) = DulMessageWArg (type,path,name,argument,timeout,argOut,status)

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

(>) type (DitsMsgType) The type of message to send. One of

DITS_MSG_KICK	Kick an action
DITS_MSG_OBEY	Start an action
DITS_MSG_GETPARAM	Get a parameter value
DITS_MSG_SETPARAM	Set a parameter value
DITS_MSG_CONTROL	Dits control message
DITS_MSG_MONITOR	Dits parameter monitor message

See DitsInitiateMessage for full details of each message type.

(>) path (DitsPathType *) The path to the task as returned by DitsGetPath or DulGetPathW.

(>) name (char *) The name associated with the message. Depends on message type but for OBEY and KICK, this is the action name.

(>) argument (SdsIdType) An argument to the message. This should be an Sds id. See DitsGetArgument and DitsPutArgument for more details on action arguments.

(>) timeout (double) If positive, this is a timeout to apply to the obey operation.

(<) argOut (SdsIdType *) The returned argument is written here. If a null argument is supplied, the behaviour is as per DulMessageW(). Otherwise, the SDS ID of any output argument associated with the reply is written here. The caller will be responsible for calling SdsDelete() and SdsFreeId() on this.

(!) status (StatusType *) Modified status.

Include files: Dul.h

Prior requirements: Dits must have been initialised.

See Also: Dul library document, DulGetPathW(3), DulLoadW(3), DitsInitiateMessage(3), DitsActionWait(3), DitsUfaceWait(3), DulMessageW(3), Dits specification.

Support: Tony Farrell, AAO

A.22 DulParseFileName — Parse a file specification using defaults and search paths.

Function: Parse a file specification using defaults and search paths.

Description: This routine is meant to parse file specifications using defaults and search paths in order to return the name of a file to be created.

Two file specifications are supplied, the "filespec" to find and a "defaultspec". Any components not supplied as part of the filename are taken from corresponding parts of defaultspec.

File specifications are made up of one of the following formats

```
DIRECTORY_SPEC:name.type
DIRECTORY_SPEC/name.type
```

There are considered to be three components . The first component is the "DIRECTORY_SPEC", which specifies the directory in which to find the file. If neither the "filespec" or "defaultspec" specify a "DIRECTORY_SPEC", then the current default directory is used.

The second component is "name". This consists of the string between the the last slash (or the colon) and the last period. If not supplied in "defaultspec", then "filespec" must supply this component.

The third component is "type". This consists of the string between the the last period and the end of the string. If not supplied in either "filespec" or "defaultspec", then nothing is supplied. Note that if this component is supplied under Unix/VxWorks in "defaultspec", then it is not possible to select a file which does not have a type. This is required to get the defaulting effect required in most cases.

Both "defaultspec" and "filespec" as first processed through DulTranslate to translate environment variables or logical names represented using formats accepted by DulTranslate. See DulTranslate for more details.

Unix/VxWorks Notes: If the directory specification ends in a colon then "DIRECTORY_SPEC" is considered to be an environment variable which will be translated. The result of this translation may be a colon separated list of directories. The resultant file specification will be in the first of these directories. For these operating systems, you may also specify this colon separated list directly.

VMS Notes: No longer supported (April 2018).

Language: C

Call:

(Void) = DulParseFileName (fileSpec,defaultSpec,flags,resultSize, result,status);

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

(>) fileSpec (char *) The specification of the file to find. See above for details.

(>) defaultSpec (char *) The default specifications for the varoius components not supplied in in fileSpec. For components are not supplied here, the corresponding components in current_directory:*.* is used under VMS and current_directory:* under Unix/VxWorks. This argument is optional and my be specified as a null pointer.

(>) flags (int) A mask of flags. Currently, none are defined. Just set to 0.

(>) resultSize (unsigned int) Size of the result string

(<) result (char *) The resultant file specification.

(!) status (StatusType *) Modified status. Will be DUL__STRLEN if the result string is too small. Other errors are possible from subsidiary routines.

Include files: DulFindFile.h

See Also: Dul library document, DulFindFile(3), dparsefile(1).

Support: Tony Farrell, AAO

A.23 DulReadFacility — Reads a message facility table (_msgt.h) file

Function: Reads a message facility table (_msgt.h) file and dynamically adds the facility to the current program.

Description: The message facility table file is a C include file is generated by the messgen program and defines a couple of structures which are normally passed directly to MessPutFacility. This normally means all facilities must be known at compile type. This function reads such an include file and creates the required structures dynamically. It then makes this facility known using MessPutFacility This allows facilities not known at compile time to be added.

Language: C

Call:

(void) = DulReadFacility (filename, status)

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

(>) filename (const char *) Name of the message facility table file.

(>) status (StatusType *) Modified status

Include File: dul.h

External functions used:

MessPutFacility	Mess	Put a facility
ErsRep	Ers	Report an error
strlen	Crtl	return the length of a string
strcpy	Crtl	copy one string to another
malloc	Crtl	allocate memory
fopen	Crtl	Open a file
fgets	Crtl	Read a line from a file
ftell	Crtl	Return the current position in a file
fseek	Crtl	Goto a positon in a file
fclose	Crtl	Close a file

See Also: Dul library document, MessPutFacility(3), Mess package spec.

Support: Tony Farrell, AAO

A.24 DulTranslate — Translate environment variables and logical names in strings

Function: Translate environment variables and logical names in strings using the Unix shell style.

Description: Given a string, translate any environment variables and return the result. The standard formats normally accepted by unix shells, such as `$NAME`, are accepted. When the former form is used, the name is terminated by any character except a period, comma, underscore, digit or alphabetic character.

Language C

Call:

(void) = DulTranslate (string, outlen, outstr, status)

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

(>) string (const char *) The string to translate

(>) outlen (int) Length of output buffer - outstr.

(<) outstr (char *) Place to write the translated string.

(!) status (StatusType *) Modified status

Include File: dul.h

External functions used:

isalpha	`CRTL`	Indicates if a character is alphabetic.
isdigit	`CRTL`	Indicates if a character is numeric.
malloc	`CRTL`	Allocate memory.
free	`CRTL`	Free memory.
strlen	`CRTL`	Return the length of a string.
getenv	`CRTL`	Return the value of an environment variable/

External values used: None

Prior requirements: None

See Also: Dul library document, DulParseFileName(3), DulFindFile(3).

Support: Tony Farrell, `AAO`

B Programs

This section details various programs avaiable in the Dul sub-system.

B.1 xditscmd — A simple X Windows based interface to DRAMA.

Function: A simple X Windows based interface to DRAMA.

Synopsis: xditscmd [xt options] [options] [task[@node]]

Under VMS, individual strings should be enclosed in quotes to preserve case (the default is to convert to lower case).

Description: This program provides a X-Windows/Motif user interface to a Dits tasks.

In addition to displaying the results of normal **Dits** commands, this program will display any image structures returned by commands in a popup image window.

Command arguments:

task The initial name of the task to send the message to. If a remote task use the format task@node, where node is the internet node name of the machine the task is running on. This value can also be set or be changed once the program is running.

Options: Only the -b and -n options are not settable after the program is running.

xt options Any options accepted by the X-Windows toolkit. See XtAppInitialise(3) for details.

-b size Size is the total message buffer size. Default 615000 This size should be about 20% bigger then that required by the values specified by the -m option. This option cannot currently be set once the program is running.

-m n1:n2:n3 Sets message buffer sizes for the connection.

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 1000:600000:1 which should be sufficient for most tasks. This value can also be set from the options pulldown, buffer menu.

-n name The name this task should register itself as. Default is the name of the top level widget (which defaults to Xditscmd but can be set by the -name Xt option). This option may be required if you intend to have several versions of this program outstanding at one time.

-o Send obey message (default).

-k Send kick message.

-s Send set message.

-G Send get message.

-c Send control message.

-p Send monitor message.

-z When supplied, the task name and message type are set using the options pull down, configuration item instead of on main screen. This reduces the space used on the screen.

-f list Specifies a space separated list which is a list of message facility files (_msgt.h files). These facilities are made known to this program. These facilities are in addition to those added by the facilities resource.

Main Window: The Menu Bar contains a File Menu, Commands Menu and Options Menu. The file menu contains the "Exit" button which will cause the program to exit.

The commands menu contains three items- Lose Path => This button which will cause the path to the current task to be closed. Delete Task(polite) => Causes the current task to be deleted in a polite way. Delete Task(force) => Causes the current task to be delete forcefully.

The Options menu contains four or five items-

Buffers	Pop up the buffers dialog which allows you to set buffer sizes
Configuration	Only present if the "-z" option was selected. It Pops up the configuration dialog allowing you to set the task name, node name and message type.
Miscellaneous	Pop up the miscellaneous dialog which allows you to set case conversion options and timeouts on various operations.
Load	Pop up the load program dialog which allows you to load a program using the imp master task. Image Test => Tests the imaging system by creating a test image and displaying it.

The first item on the main window is a scrolled list of messages output by Dits.

Next we have the optional configuration section. If the "-z" option was supplied, this is instead available in the "options" pull down, "configuration" item. It allows you to set the task name, node name and message type for the next message to be sent. The clear button causes the node and task name entries to be cleared.

Next we have the command area. First a list of previous commands is displayed, followed by the command entry area where you can type commands (action names followed by argument lists). The arrow keys can be used to scroll back through the list of commands. Hitting return will cause the current command to be executed. You can have multiple commands to one or more tasks outstanding at any one time.

Imager Window: The imager window is a simple image display system. It is displayed whenever a message from Dits contains an image structure as its argument. Note that it takes some time to scale and display an image.

The Menu Bar contains a File Menu and Options Menu. The file menu contains the "Close" button which will cause the imager window to be closed (releasing resources) and the "Refresh" button, which rescales and rewrites the image. The file menu also contains items which allow you to write the current image to a file and retrieve it from a file (using SdsWrite()/SdsRead())

The Options menu contains "Scaling" button, which activates the Scaling popup. This allows you to set the scaling of the image. The "From Data" button sets the scaling to the image low and high values. A high value of zero causes the high value to be set to the data high value automatically on each redisplay. The scaling low and high values can be any double size floating point values.

The imager window scroll bars allow you to position the image (which is not zoomed either in or out) within the output window. If the window size exceeds the image size, the scroll bars are ignored.

Clicking a mouse button anywhere in the image causes the position within the image and the pixel value at the position, to be displayed at the top of the window.

Note that if you iconify the imager window, it will be automatically deiconified when a message containing an image structure is received.

When you are finished with the imager window, release resources by closing the window using either the file menu close button or the window manager close button. Performance will be improved if the server enables backing store by default.

X defaults: `XDITSCMD` uses the standard X-Windows Toolkit resource system. Fallback resources are provided to ensure the program works even if a resource file is not found. The file "xditscmd.ad" is the source of the fallback resources and can be used as a guide for user modifications. It is found in the `DUL_DIR` directory.

To create a user specific resources, create a file named Xditscmd and set the environment variable `XUSERFILESEARCHPATH` to the value "directory/`%N`" where "directory" is the directory containing this file. (Under `VMS`, create a file named "xditscmd.dat" in the directory pointed to by the logical name "`DECW$USER_DEFAULTS`"

Of particular interest are the application specific resources (those not related to a particular widget). The available values are

nodeName A text string which specifies the default node which the target task resides on or the node on which to load the specified program (load dialog). Default is the current node.

taskName A text string which specifies the name of the target task. No default is supplied.

program A text string which specifies the name of the program to load (load dialog). No default is supplied.

processName A text string specifing the name to be given to the created process (load dialog). No default is supplied.

logFile If logging is to be written to a log file, the name of the log file. The default is xditscmd.log

globalBufferSize The size of the global buffer for Xditscmd. See DitsInit() for details (buffers dialog). Default is 605000.

messageSize An unsigned integer. The Normal size of messages sent to other tasks. See DitsGetPath() for details. Default is 1000.

maxMessages An unsigned integer. The maximum number of messages to be queued. See "DitsGetPath()" for details (buffers dialog). Default is 1.

ReplySize An unsigned integer. The normal size of messages sent from other tasks to this task. See DitsGetPath() for details. (buffers dialog) Default is 600000.

maxReplies An unsigned integer. The maximum number of replies to be queued. See DitsGetPath() for details (buffers dialog). Default is 1.

pathTimeout An unsigned integer. If timeouts are enabled on get path operations then this is the the default timeout (misc dialog). Default 100.

actionTimeout An unsigned integer. If timeouts are enabled on action messages then this is the the default timeout (misc dialog). Default 100.

taskNodeCase One of the values `UPPER`, `LOWER` or `NONE` indciating the case conversion to be applied to task and node names (misc dialog). Default is `NONE`.

actionCase One of the values `UPPER`, `LOWER` or `NONE` indciating the case conversion to be applied to action names (misc dialog). Default is `NONE`.

logging A logical value indicating if logging is to be enabled (misc dialog). Default is False.

logtoFile A logical value indicating that if logging is enable, then it should be copied to the log files specified by the logFile resource value. (misc dialog). Default is False.

pathTimeoutEnable A logical to enable or disable timeouts on get path operations (misc dialog). Default is False.

actionTimeoutEnable A logical to enable or disable timeouts on normal messages (misc dialog). Default is False.

messageType Indicates the initial message type (main window/config dialog). One of `OBEY`, `KICK`, `SET`, `GET`, `CONTROL` or `MONITOR`. Default is `OBEY`.

configDialogEnable Indicates if the config dialog should be enabled. If False, then items normally displayed in the config dialog are displayed on the main menu. If true, the config dialog button is added to the options menu. If False, then it can be overriden by the -z option. Default False.

facilities A space separated list of message table (_msgt.h) files to be loaded on startup.

See Also: ditscmd(1), XtAppInitialise(3), Dits specification, X-Windows documentation, X-Windows toolkit documentation, Motif documentation

Language: C

Support: Tony Farrell, `AAO`

B.2 dfindfile — Find a file using search paths, defaults and wildcards.

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

Synopsis: dfindfile [options] file [file...]

Description: This program will return the a name which can be used to open given files, after the application of defaults, wildcards and search paths. For example,

dfindfile "`FRED_DIR`:*.c"

dfindfile -default "*.c" `FRED_DIR`:

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 directorysearch 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 shell.

Options;

-default name Specifies defaults file name components.

-oneonly Return only the first file found. By default, all files which match the specification are returned.

-nowild Disallow wildcards.

-impdir If a directory is specified using the `<dir>:<file>` approach, with `<dir>` an environment variable name, then return the original `<dir>` specification after confirming it can be translated. This allows us to determine a directory spec appropriate IMP_Master to use. If you don't use this flag, you may find IMP_Master has problems with the length of the translated environment variable.

See Also: Dul library document, dparsefile(1), DulFindFile(3).

Author: Tony Farrell, `AAO`

B.3 dparsefile — Parse a file name using defaults and search paths.

Function: Parse a file name using defaults and search paths.

Synopsis: dparsefile [options] file [file...]

Description: This program will return the a name which can be used to create the given files, 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 directorysearch 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 the shell.

Options;

-default name Specifies defaults file name components.

See Also: Dul library document, dfindfile(1), DulParseFile(3).

Author: Tony Farrell, `AAO`

References

[1] Tony Farrell, AAO *05-Oct-1994, DRAMA C++ Interface.* Anglo-Australian Observatory Draft **DRAMA** Software document.

[2] Tony Farrell, AAO. *18-Feb-1993, A portable Message Code System.* Anglo-Australian Observatory **DRAMA** Software Document number 6.

[3] Tony Farrell, AAO. *29-Apr-1995, Dits Specification* Anglo-Australian Observatory **DRAMA** Software Document number 5.