ANGLO-AUSTRALIAN OBSERVATORY

AAO Software Report 107
2dfdr Maintenance Guide

Jeremy Bailey, Tony Farrell, Ron Heald
25 August 2005

2dfdr Maintenance Guide

Contents

Revisions:

01-Jul-2003:	TJF. Converted original document 2nd June version from Latex to MS Word. Previously did not have an ASD number or version number. When changes were complete, this became ACMM version 1.3.
03-Jul-2003:	TJF. Modified to describe use of acmmSystemBuild to do build.
26-Feb-2004:	RWH Bring up to current development, Fix typos.
02-Mar-2004:	RWH Another iteration.
17-May-2004:	RWH Improve Execution Task section.
28-Jan-2004:	RWH More work on drcontrol and drexec sections.

1 2dfdr Overview

2dfdr is the AAO's data reduction software for fibre feed Multi-Object spectrographs. It was originally written for the 2dF instrument, but has been modified to support other similar instruments including 6dF, Spiral, and LDSS. It is currently being modified to support the new AAOmega instrument.

2dfdr uses several languages and tools to accomplish its function. It uses the compiled languages Fortran, C, and C++, and the Tcl scripting language. To provide the GUI, it uses the Tk graphical interface development tool, and the Tix set of user interface components that are built on Tk. It also uses AAO's DRAMA distributed information system (middleware) for communication between tasks, and the Starlink CNF and F77 Mixed Language Programming – Fortran and C package.

1.1 Glossary

Term	Description
ACMM	AAO's source code management system
DRAMA	AAO's distributed real-time application programming environment (middleware) used by 2dfdr to support multitasking communications in portable way
FITS	Flexible Image Transport System data file format
GNS	???
GSN	???
GUI	Graphical User Interface
HDS	Starlink's Hierarchical Data System
NDF	Starlink's Extensible N-Dimensional data file Format
SDS	AAO's Self defining Data Structures
Starlink	The organization charged with supporting the computing needs of the U.K. Astronomy community. Also used to refer to the suite of software released by Starlink.
Tcl	Tool Command Language, a scripting language
Tk	Tcl's windowing toolkit used to build GUI's
Tix	An extended set of GUI widgets for Tk

2 Building 2dfdr

2dfdr is stored in and built using the AAO's source code management system – known as ACMM. [1].

2.1 Prerequisites

The following prerequisites are required to build ACMM.

- A machine with a supported architecture, currently Solaris 8 and 9, and Linux RedHat 7.3 and 9 (normally the architectures supported by Starlink).
- Sufficient disk space (~350 MB)
- C, C++ and Fortran compilers.
- A recent version of DRAMA with some changes to the DramaConfig sub-system (ACMM version 3.9 of the DramaConfig sub-system). This is presumed to be available in directory `~drama` (if it is elsewhere, then replace `~drama` by the directory containing DRAMA).
- A recent version of Starlink (Spring 2003, aka Summer 2003, was used as of this writing) in /star. 2dfdr uses a number of Starlink subroutine libraries. To build 2dfdr the Starlink software collection must be installed and must be accessible via a link to /star. The Starlink software is available on CD from Starlink (see http://www.starlink.rl.ac.uk).
- Access to ACMM. See section 2.6

2.2 2dfdr basic build procedure

This depends a little on the architecture you are using; through we hope to change this in the future.

On Solaris

```
~drama/dramastart[2]
acmmSystemBuild 2dfdr
```

On Linux

```
~drama/dramastart[2]
acmmSystemBuild 2dfdr  last -L/usr/X11R6/lib[3]
```

[1] ACMM is based on the ESO CMM software, used with permission by the AAO.

[2] There may be an appropriate version specification required. At AAO Epping as of December 2004, you need to add "`-v 14dec04_linux`" to the command.

The result of these commands is the fetching of all the software from ACMM into subdirectories, the building of the software and creation of two extra sub-directories. The first of these is named “release”, and contains the full set of files (libraries etc) created by the build. It is the install directory for the various sub-systems being built.

The second extra sub-directory is named `2dfdr-{arch}`, where {arch} will be either “`sun4_solaris`” or “`linux_x86`” indicating the build target architecture. This directory contains an independent release directory. It can be tar’ed up and moved at will (although there may be some restrictions on operating system versions due to sharable library version issues). At run time, the environment variable DRCONTROL_DIR should be defined to this directory. (Note the location should be such that the value of DRCONTROL_DIR is less then 40 characters, due to an apparent bug in Starlink).

A log of the build is written to the “install.log” file. You can examine the contents of this file to help determine build problems. It is often useful to open another terminal window and use “tail –f install.log” during the build process.

2.3 Partial Builds

As above, but you may want to add the following flags before “`2dfdr`.”

Flag	Description
`-start {subsystem}`	Start building from the specified sub-system (as mentioned below).
`-stop {subsystem}`	Stop building after the specified sub-system is built. Note this prevents the post build hook (used to create the release directory) from being built.
`-nofetch`	Don’t check if the source directories are up to date with those in the source archive.

2.4 The Subsystems

The following table lists the sub-systems fetched from ACMM and used in the 2dfdr build. The sub-system name indicates the ACMM name and the sub-directory in which the software will be placed. The order in the table is the build order.

[3] The requirement for “last –L/usr/X11R6/lib” is to work around a problem in the DRAMA configuration files. It is needed to correctly locate the X11 libraries. A future revision to the DRAMA configuration files will avoid this issue. Note that the current (Jul 03) location of Linux DRAMA is `/instsoft/dramasrc/linux/drama-v1.4.2/`, which must replace `~drama` in this line. Also note that “last” is the normal default and is only needed here to allow the following argument.

Subsystem	Description
2dfdr*	Contains the 2dfdr description used by the `acmmSystemBuild` and `acmmSystemFetch` commands. The file 2dfdr.cfg contains this information.
Tcl	The Tool Command Language. The version must match that used by Starlink – which can be determined by running /star/bin/wish* and executing the command "info tclversion".
Tk	Tcl's Windowing toolkit. Details as per Tcl. From Version 8.0 of Tcl and Tk use the same version numbers. Prior to this, the version numbers are different. 2dfdr as supported by acmmSystemBuild does not work with versions of Tcl/Tk prior to version 8.0.
Tix	A Tck/Tk widget set used by 2dfdr. The version of Tix used must support the version of Tcl used. Note that the "THEVER" specification in 2dfdr.cfg must specify the version of Tcl/Tk to be used.
wcstools	World Coordinate system tools. A set of FITS header tools. Only the `gethead` and `sethead` programs are needed.
sds	The DRAMA Self Defining Data system. This must be rebuilt against the version of the Tcl/Tk being used. (Also, the exact version of the C++ compiler and Fortran compiler).
DramaDul	The DRAMA Utilities Library (DUL). This must be rebuilt against the version of the Tcl/Tk being used. (Also, the exact version of the C++ compiler and Fortran compiler).
DramaTcl	The DRAMA Tcl Interface (DTCL). This must be rebuilt against the version of the Tcl/Tk being used.
aaoDramaTclUtil	A set of AAO DRAMA Tcl scripts.
clalib*	AAO data reduction Fortran "class" library.
nswc	The (public domain) Naval Surface Warfare Center library of mathematical routines.
tdfred*	2dF data reduction Fortran "class" library.
drexec*	The 2dfdr data reduction execution task.
drcontrol*	The 2dfdr data reduction control program. This also contains the release script and run time start up scripts.
ASD107*	Contains this document and the 2dfdr User Manual. Nothing is actually built in this sub-system.

Sub-systems marked with * are the components of 2dfdr itself. The remaining sub-systems are dependencies.

2.5 Future changes to build system

This section notes intended future changes to the release and build procedure.

2.5.1 2dfdr

The build procedure needs to check the Tcl and Tk versions used are compatible with those used by Starlink. This can be done in the post fetch or pre build hooks.

A way is needed to extract the acmmSystem files to allow a build without having access to the ACMM library.

A way is needed to extract the definitions needed to build the individual components (the post build hook could extract the environment variable values into a file).

2.5.2 DRAMA

DRAMA must be changed such that instead of having to override the values of the SYSTARGET, SYSHOST and SYSRELEASE, only one value needs to be overridden.

The DRAMA configuration files must be changed such that we can specify the location of Tcl/Tk independently of specifying the locations of X11 and other things required to link Tcl/Tk.

2.6 ACCM

2.6.1 Setting Up for ACMM

ACMM is the AAO source code archiving system. In order to use ACMM certain files and environment variables must be set appropriately. If the command "acmmWho" returns an error indicating an unknown command, then the setup is not correct. To do so, execute the setup line from the following table according to your location and shell. This line can be added to your shell start up script so that ACMM commands are always available.

Location	Shell	ACMM Set up Line
Epping Solaris	sh/zsh/bash/ksh	`. /local/soft/solaris/acmmlib/acmm_setup.sh`
Epping Solaris	Csh/tcsh	`source /local/soft/solaris/acmmlib/acmm_setup.csh`
Epping Linux	zsh/bash/ksh	`. /local/soft/linux/acmmlib/acmm_setup.sh`
Epping Linux	Csh/tcsh	`source /local/soft/solaris/acmmlib/acmm_setup.csh`
Coona Solaris	sh/zsh/bash/ksh	`. /instsoft/acmm/acmmlib/acmm_setup.sh`
Coon Solaris	Csh/tcsh	`source /instsoft/acmm/acmmlib/acmm_setup.csh`

If “acmmWho” indicates that a file named “.acmmrc” can’t be opened for reading, then you need to arrange access to ACMM. Please send an e-mail to tjf@aaoepp.aao.gov.au.

2.6.2 Using ACMM

Sufficient knowledge to build 2dfdr has now been provided. If you need to modify the 2dfdr archived source code, then you must become familiar with the basic operation of ACMM. This is described in AAO Software Document 101. Type “acmmCopy ASD101” to fetch this document. This will create a local directory containing this document. It also contains a PowerPoint presentation on ACMM. Note that the basic operations require only 4 commands (acmmCopy, acmmModify, acmmArchive, and acmmCheckForArchive) in addition to acmmSystemBuild.

2.7 Modifying 2dfdr

There are three ways of modifying 2dfdr. Note that it is expected that most people modifying 2dfdr will only need to change tdfred, drcontrol, drexec, and just possibly clalib.

2.7.1 Modifying the Archive

This should only be done with the intention of introducing changes into the main code. It should be done using the appropriate ACMM commands by an authorized maintainer. See the ACMM document (ASD101) for details.

2.7.2 Branching

It may be appropriate to branch 2dfdr for experiential work or to provide bug fixes. ACMM supports branching but it is intended that braches either die or are quickly merged back into the main-line code. See the ACMM document (ASD101) for details.

2.7.3 Modifying the Readonly Version

This is a dangerous technique due to the chance of the changes getting lost. In a normal build, all the source files are marked “readonly”. You can change this using “chmod u+w {file}” and then make changes to the file. This may be appropriate for experimental changes by someone without access to ACMM, but they must keep careful track of the changes and ensure that if they are intended for the main-line code, they get into the ACMM version at some point.

2.8 Build Configuration

The order of the build and options to the build are set in the 2dfdr subsystem, file 2dfdr.cfg. This obeys the rules laid down by acmmSystemFetch and acmmSystemCapture. The best source for information about this scheme is the PowerPoint presentation in the ASD101 ACMM module.

2.8.1 Setting Compiler Options

When modifying the 2dfdr.cfg file or running dmkmf independently (not yet practical) you should set compiler options (e.g. Optimize level or debug) by running `dmkmf` with a `-O` or `-g` argument. The `-O` option should normally be used with the nswc, `clalib` and `tdfred` libraries to get good execution speed. With the other sub systems it is not critical.

2.8.2 System Version Capture

The ACMM build configuration file (2dfdr.cfg) allows the ACMM version of each sub-system to be specified. When specified as "last" the last available version is used, but it is often appropriate to specific particular versions for all sub-system so that you can return to a known configuration.

ACMM supports the idea of capturing the current version of each sub-system into the system configuration file (2dfdr.cfg). This is done with the acmmSytemCapture command. But this is not necessarily appropriate for Tcl/Tk/Tix. Here you must specify the version appropriate for the version of Starlink to be used. So if you used acmmSystemCapture, you must unwind any change to the specifications for Tcl/Tk/Tix.

⚠ **acmmSystemCapture may mess up the Tcl/Tk/Tix version specification.**
You must hand-edit 2dfdr.cfg after running acmmSystemCapture to check and if necessary fix this.

2.9 Version Number and Date

The 2dfdr version number and date (which appears in the title bar of the user interface) are set in the drcontrol and drexec tasks. The following lines (or something similar) will be found in the drcontrol.C and drexec.c files:

```
extern char* DrcontrolVersion;
extern char* DrcontrolDate;
```

Compiling the files drcontrolversion.c and drexecversion.c sets these variables. This is done in such a way to force the file to be recompiled if the version number changes. The version number is based on the ACMM version number and is therefore updated automatically when the subsystems are updated.

2.10 Running 2dfdr after a build

To run the 2dfdr version you have just built, do the following:

```
setenv DRCONTROL_DIR {build-directory}
source $DRCONTROL_DIR/2dfdr_setup
```

where build-directory is, for example, `/home/rwh/2dfdr/2dfdr-linux_x86` or `/home/rwh/2dfdr/2dfdr-sun4_solaris`. The command "drcontrol" should now start up 2dfdr.

⚠	**Due to a problem in the Starlink libraries used by 2dfdr, the value for DRCONTROL_DIR must not exceed 40 characters.** Getting this wrong will cause the plotting facilities to fail.

3 Testing 2dfdr

There is a 2dF sample data set available which can be used to test the basic functionality of 2dfdr. It can be found at

ftp://ftp.aao.gov.au/pub/2df/sample_data.tar.Z

It contains the following files:

11may0001.sdf
11may0002.sdf
11may0003.sdf
11may0004.sdf
11may0005.sdf
README

To run 2dfdr on this dataset untar the data, `cd` to the directory containing the data and type the following.

```
drcontrol&
```

The 2dfdr user interface and two plot windows should then appear. Click the SkySub tab on the left section of the screen and select “Throughput Calibration Method - SKYFLUX(COR)”.

In the automatic reduction section on the top right of the user interface, click the **Setup** button, and select **OK** in the dialog box which appears. 2dfdr should then show 5 files in the automatic reduction box.

Click the **Start** button in the automatic reduction section and it will reduce the data going through the five files in turn using the default TRAM method.

To repeat the reduction exit from 2dfdr and type:

```
rm *red.sdf *tlm.sdf
```

And start drcontrol again. This time select the Extract tab, select Method: FIT and turn on "Subtract Scattered Light". Click the SkySub tab and select “Throughput Calibration Method - SKYFLUX(COR)”.

Click **Setup** and **Start** as before. The reduction proceeds as before but take a lot longer in this mode

4 The DRAMA Tasks drcontrol and drexec

4.1 Introduction

The 2dF data reduction system makes use of two DRAMA tasks, the user interface task (drcontrol) and the execution task (drexec). The drcontrol task does its work by requesting DRAMA actions from one or more execution tasks.

The drcontrol main program is written in C++. drcontrol also makes extensive use of Tcl/Tk and the Tix widget set to provide a GUI.

The drexec main program is written in the C language, while the actual data reduction algorithms are written in Fortran in the form of a class library (see Sections 5 and 6). The drexec task is linked with the class library and calls the Fortran routines by making use of the Starlink CNF/F77 C language to Fortran language package (described in SUN/209). In essence drexec provides a DRAMA interface to the Fortran class library interface described in the clalib and tdfred sections.

4.2 drcontrol

The drcontrol task "unit of work" is the reduction of a single file. A work unit is passed by the drcontrol task to the drexec task in the form of a single EXECUTE action. The drexec task opens all files needed in the course of the reduction and closes all files when it is complete.

Notice drcontrol handles both NDF and FITS format files, converting FITS to NDF format before passing them to drexec. Only NDF format files are seen by drexec.

4.3 drexec

4.3.1 The EXECUTE action

The drexec task does most of its work in response to a DRAMA action with the name EXECUTE. The EXECUTE action specifies, in its argument structure, the data reduction operation to be performed. The operation can range from execution of a single step to an arbitrarily complex sequence of steps.

Operations using the class library methods require specification of the object to be operated on, the method to be invoked, and the arguments to be supplied to the method (in the form of a SDS arguments structure). The class library operates on objects specified by means of their NDF identifiers (described in SUN/33). Since NDF identifiers are only meaningful within the context of a single process, the EXECUTE action has to be told how to derive these identifiers from container files. This information is provided by means of the FILES section in the EXECUTE action argument structure. The sequence of methods to be executed is specified in the SEQUENCE section of the structure.

4.3.2 The FILES component

The FILES component of the argument structure links parameter names with file names and specifies how an NDF identifier is to be obtained from the file. It is an SDS structure with the following components:

DIRECTORY	The DIRECTORY component is a character string specifying the directory for all files. The drexec task sets its default to this directory before accessing any files. Thus all files will be taken from this directory.
Parameter Entries	Subsequent components in this structure are parameter entries with the structure component name being the SDS parameter name. The drcontrol task normally creates parameter entries with names of the form P0, P1, P2 etc. allocated sequentially. However, any valid SDS name will be accepted by drexec as a parameter name.

Each parameter entry is itself a structure and must have a NAME component specifying the name of the file (with any extension) to be used, and an ACCESS component specifying how an NDF identifier is to be obtained from the filename. ACCESS must be one of the following strings:

READ	The file will be opened for READ access, by using the READ method of the NDF class.
UPDATE	The file will be opened for UPDATE access, by using the UPDATE method of the NDF class.
TEMPLATE	A new file will be created based on a "template" file, using the template method of the NDF class. The new file contains everything in the template file except for the DATA, QUALITY, and VARIANCE components. An additional argument, TEMPLATE, is required with this access to specify the template file. This should be a parameter reference beginning with a $ as described below. Two optional arguments, ADD_SUFFIX and REMOVE_SUFFIX, can also be specified.
NEW	A new identifier will be created by the method to be invoked in the sequence section. No identifier need exist before invoking the method.

Notice all files are closed when the EXECUTE action completes.

4.3.3 The SEQUENCE component

The SEQUENCE component is a one dimensional structure array with one component for each step in the sequence of operations to be executed. Each step results in the invocation of one method in the class library. The structure for each step contains the following components:

OBJECT	The object is specified as a character string containing the parameter entry name of the object to be operated on prefixed by a $ character. The same name must be present in the FILES component.
METHOD	The name of the method to be invoked.
CLASS	The name of the class for which the method should be invoked. This is optional and normally unnecessary. When CLASS is not specified the method will be invoked by using the generic class routine CLA_GENER which will determine the appropriate class by looking at the object itself. Specifying the class explicitly will override this choice. Currently the CLASS item is only suported for the GROUP class.
ARGS	The arguments to be passed to the class. This is identical to the arguments structure which would be passed to the class library routines, except for arguments which are NDF identifiers. These are specified via parameters, using a character string with a $ character preceding the parameter entry name. The parameter name is replaced by the corresponding NDF identifier before the arguments structure is passed to the class routine.

4.3.4 Examples of EXECUTE argument structure

This example structure executes the PLOT method on the file /data4/jab/tests/im256.sdf to generate a false colour plot on device `xwindows'.

```
Arguments            Struct
   FILES                Struct
      DIRECTORY            Char   [17] "/data4/jab/tests"
      P0                   Struct
         NAME                 Char   [10] "im256.sdf"
         ACCESS               Char   [5] "READ"
   SEQUENCE          Struct  [1]
      SEQUENCE[1]       Struct
         OBJECT            Char   [4] "$P0"
         METHOD            Char   [5] "PLOT"
         ARGS              Struct
            DEVICE            Char   [9] "xwindows"
            INTERACTIVE       Int    1
            PLOTTYPE          Char   [7] "COLOUR"
```

The following example constructs a group object containing five files.

```
Arguments           Struct
   FILES               Struct
      DIRECTORY            Char   [17] "/data4/jab/tests"
      P0                   Struct
         NAME                 Char   [12] "test_group"
         ACCESS               Char   [4]  "NEW"
      P1                   Struct
         NAME                 Char   [7] "r2.sdf"
         ACCESS               Char   [5] "READ"
      P2                   Struct
         NAME                 Char   [7] "r3.sdf"
         ACCESS               Char   [5] "READ"
      P3                   Struct
         NAME                 Char   [7] "r4.sdf"
         ACCESS               Char   [5] "READ"
      P4                   Struct
         NAME                 Char   [7] "r5.sdf"
         ACCESS               Char   [5] "READ"
      P5                   Struct
         NAME                 Char   [7] "r6.sdf"
         ACCESS               Char   [5] "READ"
   SEQUENCE             Struct  [1]
      SEQUENCE[1]          Struct
         METHOD               Char   [11] "MAKE_GROUP"
         OBJECT               Char   [4] "$P0"
         CLASS                Char   [6] "GROUP"
         ARGS                 Struct
            FILENAME             Char   [12] "test_group"
            PERMANENT            Int    1
            NOBJECTS             Int    5
            OBJECT1              Char   [4] "$P1"
            OBJECT2              Char   [4] "$P2"
            OBJECT3              Char   [4] "$P3"
            OBJECT4              Char   [4] "$P4"
            OBJECT5              Char   [4] "$P5"
```

The following example adds the constant 0.1 to the file im1024.sdf to give the output file junk.sdf.

```
Arguments            Struct
   FILES                Struct
      DIRECTORY            Char   [17] "/data4/jab/tests"
      P1                   Struct
         NAME                 Char   [11] "im1024.sdf"
         ACCESS               Char   [5] "READ"
      P0                   Struct
         NAME                 Char   [5] "junk"
         ACCESS               Char   [9] "TEMPLATE"
         TEMPLATE             Char   [4] "$P1"
   SEQUENCE             Struct [1]
      SEQUENCE[1]          Struct
         METHOD               Char   [13] "CARITHMETIC3"
         OBJECT               Char   [4] "$P0"
         ARGS                 Struct
            OBJECT1              Char   [4] "$P1"
            CONSTANT             Float  0.1
            OPERATION            Char   [2] "+"
```

5 The Clalib and Tdfred Class Libraries

5.1 Summary

The data reduction class libraries consist of many Fortran subroutines that provide facilities for writing data reduction software. The subroutines operate on Starlink standard NDF data structures. The subroutines do not depend on any particular software environment and can be used in standalone Fortran programs, though they can also be used effectively within the ADAM environment. The library is written in portable Fortran 77 and currently exists in versions for SUN/Solaris and Linux architectures.

The tdfred library is meant to provide functions specific to 2dfdr, while the clalib functions provide data reduction functions for more general use. In addition to 2dfdr, the clalib library is used by the "aaoplot" application maintained by Jeremy Bailey.

The organisation of the libraries attempts to use object-oriented programming methodology, and this is why they are referred to as "class" libraries. It is not necessary to have any familiarity with object-oriented programming in order to make use of the libraries.

A disadvantage to this organisation is linking is effectively done at execution time. This means a missing function will not be noticed until an attempt to call it is made. It also means that the usefulness of linking libraries is lost because all functions are loaded whether they are used or not.

This document is an introductory guide for programmers, which explains how to use the clalib and tdfred libraries to write useful programs.

5.2 Getting Started

We start by presenting a simple example of a program using the clalib class library. The example is an interactive image display program, which displays an image on a display device, and then puts up a cursor and enables a variety of operations including zooming, panning, plotting of cuts through the image etc.

The program is as follows:

```
PROGRAM IMDISP
      IMPLICIT NONE

      INCLUDE 'SAE_PAR'              ! Starlink constants

      CHARACTER*40 FILE              ! File name
      CHARACTER*40 DEVICE            ! Display device name
      INTEGER OBJECT                 ! Identifier of NDF object
      INTEGER ARGS                   ! Identifier of argument
structure
      INTEGER STATUS                 ! Status value

*  Get name of file from user
      PRINT *,'Enter name of file to display'
      READ(*,'(A)') FILE

*  Get device name from user
      PRINT *,'Enter device name'
      READ(*,'(A)') DEVICE

*  Set initial status value and start HDS
      STATUS = SAI__OK
      CALL HDS_START(STATUS)

*  Create an empty argument structure
      CALL ARG_NEW(ARGS,STATUS)

*  Get NDF ID of file
      CALL ARG_PUT0C(ARGS,'FILENAME',FILE,STATUS)
      CALL CLA_NDF(OBJECT,'READ',ARGS,STATUS)

*  Do the image display
      CALL ARG_PUT0C(ARGS,'DEVICE',DEVICE,STATUS)
      CALL ARG_PUT0L(ARGS,'INTERACTIVE',.TRUE.,STATUS)
      CALL CLA_2D(OBJECT,'PLOT',ARGS,STATUS)

      END
```

To test this a NDF file is needed, for example a file created with KAPPA or any other Starlink application that uses NDF (NDF format files normally have extension .SDF).

The program will also work on FITS files since these will be converted on the fly to temporary NDF structures.

Run the program and it will prompt for the name of the file, and then the display device name. Give it the GNS name of the display device you want to use (e.g. IKON or XWINDOWS).

This example gives an idea of the structure of a program using the class library. Most of the work is done by the two CLA_ subroutines, CLA_NDF and CLA_2D. It will be seen that both of these routines have the same calling sequence, and this is a feature of all the subroutines in the class library. The calling sequence is:

```
CALL CLA_NDF(OBJECT,METHOD,ARGS,STATUS)
```

where

OBJECT	is the NDF identifier of the object to be processed.
METHOD	is a character string specifying the operation we want to carry out on the object.
ARGS	is a structure containing the arguments which need to be supplied to carry out the operation.
STATUS	is a status argument which follows the normal Starlink inherited STATUS convention. STATUS should have the value SAI__OK on entry or the routine will do nothing. On exit the status will be SAI__OK if no errors occurred, but will be set to an appropriate error value if an error occurred. Any errors will be reported using a call to ERR_REP.

The use of the ARGS structure is probably the main difference from most Fortran subroutines. It enables routines to be both simple to use, as well as powerful and flexible. It is normally necessary to specify very few arguments in order for the routine to do its job, but there are often many more arguments which could be specified in order to provide finer control of the operation. All these optional arguments will take suitable default values if not specified in the ARGS structure.

Before calling one of the CLA_ routines, an ARGS structure must be created (by calling the routine ARG_NEW) and any necessary arguments must be loaded into the ARGS structure by calling the ARG_PUT0x routines. There is one of these routines for each of the Fortran types as follows:

Routine	**Type**
ARG_PUT0C	CHARACTER*n
ARG_PUT0D	DOUBLE PRECISION
ARG_PUT0I	INTEGER
ARG_PUT0L	LOGICAL

Routine	Type
ARG_PUT0R	REAL

In the example the same ARGS structure is reused for the two calls to the CLA_ routines. This can usually be done, but care must be taken that none of the arguments set for the first call would inadvertently change the behaviour of the second call.

Returning to the two main subroutine calls, the example program first calls CLA_NDF, specifying method READ. The READ method is the one used to read a new NDF into the system. It has a FILENAME argument which specifies the name of the file to be read. This can be the name of an NDF container file, but can also be a different type of file, since the use of *file filters* enables files of other types to be converted to NDFs. Currently FITS files are supported.

The CLA_2D subroutine is then called with the PLOT method specified. This method is used to plot a 2D image. The DEVICE argument specifies the name of the graphics device to be used. In this example the logical argument INTERACTIVE is set to TRUE, which brings the plot up in interactive mode. PLOT has many other arguments, but all of these have default values and so can be omitted in a simple example like this.

5.3 Classes and Methods

The organization of the software is into classes and methods. There is a Fortran subroutine for each class, which can be called to invoke any of the methods of that class. The above example made use of two classes, the NDF class for which the subroutine is CLA_NDF and the 2D class with subroutine CLA_2D.

The term *class* refers to the class of data object on which the methods can operate. Thus class NDF contains those operations which can be applied generally to any NDF object. Class 2D contains those operations which apply to 2 dimensional data objects.

Classes are organized into a hierarchical structure, with the most general class NDF at its root. An illustration of the class hierarchy of the library is given below.

```
        -------1D
       |
NDF---|-------2D
       |
       |-------POL-------POL2D
       |
        -------GROUP
```

Classes inherit all the methods of the classes below them in the hierarchy. Thus all the methods of the NDF class are available to any class in the hierarchy.

The way this works in practice is a call to CLA_2D specifying a method such as ARITHMETIC, which is a method of the NDF base class, will always be valid. What will happen is that CLA_2D will call CLA_NDF to handle the method. In some cases it may be necessary for the derived class to override the method of the base class and provide its own handling of an operation. For example, the ARITHMETIC method of the POL (polarization) class, would have to handle the Stokes parameters as well as the main data array.

5.4 Arithmetic Operations

The ARITHMETIC method of the NDF class handles the basic arithmetic operations (addition, subtraction, multiplication and division). These operations are not quite as straightforward as you might think, and the packaged arithmetic methods look after many little details that are often overlooked.

- Bad pixel handling - NDFs may contain pixels which are flagged as bad, and the arithmetic methods take note of these and automatically propagate them through the arithmetic operations.
- Variance propagation - If the NDFs being operated on contain variance arrays these will also be processed (on the assumption that the two operands are independent).
- Mismatched arrays - If the two NDFs have different pixel index bounds, then the operation will be performed on the overlapping region and the output NDF resized accordingly.
- Arithmetic Errors - Errors such as overflow, divide by zero etc. will be handled by setting the corresponding pixel of the output array to the bad pixel flag.

Here is an example of a program that adds two files using the ARITHMETIC method:

```
PROGRAM ADD
      IMPLICIT NONE

      INCLUDE 'SAE_PAR'              ! Starlink constants

      CHARACTER*40 FILE1             ! Input File name 1
      CHARACTER*40 FILE2             ! Input File name 2
      INTEGER OBJECT                 ! Identifier of 1st NDF
object
      INTEGER OBJ2                   ! Identifier of 2nd NDF
object
      INTEGER ARGS                   ! Identifier of argument
structure
      INTEGER STATUS                 ! Status value

*  Get name of file 1
      PRINT *,'Enter name of 1st file'
      READ(*,'(A)') FILE1
```

```
*  Get name of file 2
      PRINT *,'Enter name of 2nd file'
      READ(*,'(A)') FILE2

*  Set initial status value and start HDS
      STATUS = SAI__OK
      CALL HDS_START(STATUS)

*  Create an empty argument structure
      CALL ARG_NEW(ARGS,STATUS)

*  Get NDF ID of file 1 (opening it for update access)
      CALL ARG_PUT0C(ARGS,'FILENAME',FILE1,STATUS)
      CALL CLA_NDF(OBJECT,'UPDATE',ARGS,STATUS)

*  Get NDF ID of file 2 (opening it for read access)
      CALL ARG_PUT0C(ARGS,'FILENAME',FILE2,STATUS)
      CALL CLA_NDF(OBJ2,'READ',ARGS,STATUS)

*  Do the ADD operation
      CALL ARG_PUT0C(ARGS,'OPERATION','+',STATUS)
      CALL ARG_PUT0I(ARGS,'OBJECT2',OBJ2,STATUS)
      CALL CLA_NDF(OBJECT,'ARITHMETIC',ARGS,STATUS)

      END
```

Note that arithmetic operations such as ADD require two NDF objects, whereas the standard CLA_ calling sequence only provides for one object. The NDF identifier of the second object must be put into the argument structure using an ARG_PUT0I call.

In such cases the normal convention is that the object specified in the subroutine call is the object which will be modified by the operation, whereas the one supplied in the argument structure will be unmodified and therefore requires only read access.

Other arithmetic operations are provided by changing the OPERATION argument to "*", "– "or "/" as appropriate.

The CARITHMETIC method is similar, but provides operations involving a constant and an array. The constant is put into the argument structure.

5.5 Using the Class Library in ADAM Applications

The examples so far have been standalone Fortran programs, using Fortran I/O to obtain values of parameters from the user.

The Class Library can also be used to write ADAM Applications which use the ADAM parameter system to communicate with the user. The following example shows a simple 2D plot application written as an ADAM A-task.

```
SUBROUTINE PLOT(STATUS)
      IMPLICIT NONE
      INCLUDE 'SAE_PAR'

      INTEGER STATUS

      INTEGER OBJECT
      INTEGER ARGS

*   Get the object
      CALL NDF_ASSOC('INPUT','READ',OBJECT,STATUS)

*   Create an empty argument structure
      CALL ARG_NEW(ARGS,STATUS)

*   Get the device name and put it in the argument structure
      CALL PAR_GET0C('DEVICE',DEVICE,STATUS)
      CALL ARG_PUT0C(ARGS,'DEVICE',DEVICE,STATUS)

*   Set the interactive argument
      CALL ARG_PUT0L(ARGS,'INTERACTIVE',.TRUE.,STATUS)

*   Do the plot
      CALL CLA_2D(OBJECT,'PLOT',ARGS,STATUS)

*   Check for errors
      IF (STATUS .NE. SAI__OK) THEN
          CALL ERR_REP(' ','PLOT - Error',STATUS)
      ENDIF

      END
```

In this example the NDF_ASSOC routine is used to obtain the NDF identifier of the file to be plotted, rather than use the NDF class READ method. The DEVICE parameter is read using the ADAM parameter system, and then immediately copied into the argument structure.

5.6 Generic Operations

In the examples we have presented so far we have always used a specific class routine to perform the operation. However, there are many cases where different classes may perform the operation in different ways. An example is the PLOT method which we have already come across in the 2D class. There is also a PLOT method in the 1D class, and in the POL2D class.

We can therefore make programs more general by first determining the class of the object and then call the appropriate CLA_ routine for that class. This is a sufficiently common operation that a standard routine is provided. This routine is CLA_GENER, the generic class routine. It has exactly the same calling sequence as any of the CLA_ routines, but is not strictly a class.

To make our original IMDISP program more general we can simply replace the call to CLA_2D to a call to CLA_GENER as follows:

```
CALL CLA_GENER(OBJECT,'PLOT',ARGS,STATUS)
```

This will enable IMDISP to plot one dimensional data and polarization data as well as 2D images. Moreover, it means that in the future IMDISP will automatically acquire the ability to plot objects of any new class added to the class library, provided the class has a PLOT method provided.

CLA_GENER determines the class of an object in one of three ways. It first looks for the presence of an NDF_CLASS extension within the object. If this is present the NAME component contained in this extension specifies the class name. If no NDF_CLASS extension is found, CLA_GENER looks for a “RUNCMD” keyword name in the FITS header. If this is found, it is used to derive the object’s class. For example, if the value of the RUNCMD header is “RUN”, then the object’s class is “MFOBJECT”. If neither of the first two methods is successful, CLASS_GENER looks at the structure of the object to determine its class. For example, it looks at the dimensionality, and the presence of extensions. Thus an object with a GROUP extension is assigned to the GROUP class; a 2 dimensional object with a polarimetry extension is assigned to the POL2D class etc.

5.7 Groups of Objects

It is frequently the case in data reduction systems that an operation needs to be applied to a group of objects, rather than just to a single object. For example, if we are flat-fielding CCD frames we will probably want to apply the flat-field operation to all of the frames in a night's observation. Other cases are when we are combining a series of identical exposures.

The GROUP class supports this type of operation. A GROUP object can be created which consists of a group of individual NDFs, and operations can then be carried out on the whole group as easily as on individual objects.

A GROUP object is an NDF that contains a GROUP extension. The GROUP extension is a structure containing a list of references (using the REF package described in SUN/31) to other NDF objects. The use of references means that the NDFs forming the group can either be in the same HDS container file as the group object, or can each be an independent object in its own file.

The GROUP class provides the following methods for building and manipulating groups.

CREATE_GROUP	Create a new, empty, group object.
INSERT_OBJECT	Insert an object into a group.
DELETE_ENTRY	Delete an entry from a group.
GET_ENTRY	Find an entry in a group and return its NDF identifier.
GET_NENTRIES	Return the number of entries in a group.
DO_FOR_EACH	Invoke a specified method for every object in thegroup.

Groups may be either temporary (exist only for the duration of the program) or permanent. A temporary group may contain either temporary or permanent objects, but a permanent group may only contain permanent objects.

A Starlink Routine Modifications

A.1 The x.c File

???

A.2 The err.f file

???

B The 2dfdr release

2dfdr is distributed as self contained binary releases which contain everything needed to run on a system which may not have DRAMA, Starlink or Tcl/Tk installed. Thus most of the files in the release are the necessary support files taken from the DRAMA, Starlink, and Tcl/Tk systems.

The main restrictions on running such binary releases will be the operating system sharable libraries required. The 2dfdr build has tried to link non-sharable (static) libraries where possible, but this is not always possible. In some cases a version of 2dfdr may not run in older version of the operating system than it was built on. We have attempted to avoid this by including, where allowed, relevant sharable libraries in the release.

All these files are collected as part of the normal build by the script `drcontrol/2dfdr_release`. This script should be seen as the definition of what is required and where it comes from. The rest of this section gives more details on these files.

B.1 2dfdr specific files

The release includes the following files that are specific to 2dfdr. Note that the source here is the original source of the file. For built files, they may actually be picked up from the build release directory to ensure the last completed build is used.

File Name/Names	Source	Description
drexec	drexec directory	The drexec executable
drcontrol	drcontrol directory	The drcontrol executable
drcontrol.tcl	drcontrol directory	The drcontrol Tcl script
2dfinfo	drexec directory	The 2dfinfo executable
cleanup	$IMP_DIR	DRAMA cleanup command

File Name/Names	Source	Description
.idx files	drcontrol directory	Files which configure the user interface
.arc files	drcontrol directory	Wavelength calibration lamp data
spiral.dat	drcontrol directory	SPIRAL IFU description files
2dfdr_setup	drcontrol directory`	Setup script.
2dfdr_setup_sh	2dfdr_setup script (above)	SH shell version of 2dfdr_setup
2dfdr_run	drcontrol directory	A script used to run all 2dfdr commands. It sets appropriate environment variables before running the command. the 2dfdr_setup and 2dfdr_setup_sh scripts both execute commands via this script.
.sdf files	drcontrol/bad_pixel_masks.tar.gz	Bad pixel masks for CCDs
fibposxx.dat files	drcontrol directory.	2df slit descriptions

Normally to make a new release all that is necessary is to take an existing release directory and copy into it the new version of the `drexec` executable, and if they have changed the `drcontrol` executable and `drcontrol.tcl` file. The rest of it will usually not need changing. (But acmmSystemBuild always collects the required files).

The 2dfinfo executable is part of the `drexec` package.

B.2 Other files

The 2dfdr release inlcudes a number of files taken from the DRAMA, Starlink and Tcl/Tk systems in order to enable 2dfdr to work as a standalone system. It may be necessary to update these for new versions of the systems. These files are normally fetched by the script `drcontrol/2dfdr_release`, run as a post-build hook by acmmSystemBuild. This section gives details on these files, through the script should always be assumed to be the definition of the files which are required and the required location in the release directory.

B.3 DRAMA Files

These files will be found in the directories specified by the following environment variables:

File	Source	Description
dtcltk.tcl	$DTCL_DIR	DTCL start up script.
drama.icon	$DTCL_DIR	DRAMA Icon used by DTCL.
Dits_Err.tcl	$DITS_DIR	DITS error codes – TCL include file.
dul_err.tcl	$DUL_DIR	DUL error codes – TCL include file.

B.4 Starlink files

The following files are required for Starlink automatic conversion of FITS to NDF format, and NDF to FITS format. They are copied into the "starlink" sub-directory of the release.

```
fits2ndf

fits2ndf.ifc

ndf2fits

ndf2fits.ifc
```

The following files are required for Starlink graphics to work. They are copied into the "starlink" sub-directory of the release.

```
gks.dbs-1.37

gks.emf-1.37

gks.wdt-1.37

gns_gksdevices

gns_gksnames

grfont.dat

rgb.txt
```

These are all found in the Starlink release in /star/etc. I am not entirely sure what some of these do. The names and set of files may change in subsequent Starlink versions, but this set has worked for quite a while now.

B.5 Tcl/Tk/tix files

Tcl, Tk and Tix all require access to library script files. These are copied into the"Tcl", "Tk" and "Tix" sub-directories if the release by the 2dfdr_release script.

B.6 WCS files.

The gethead and sethead tools from the “wcstools” sub-system are copied into the release, for use by the 6df2ndf Perl script.

C Clalib Classes and Methods

Here are summary descriptions of the various classes and their methods. See the source code comments for details of the arguments used by each method.

C.1 Class NDF

Source file: class_ndf.f

This is the base class of all classes in the hierarchy and includes methods valid for any NDF file.

Methods:

ARITHMETIC	Perform arithmetic operation on the object and another specified in the argument list replacing the object with the result.
ARITHMETIC3	Perform arithmetic operation on two objects specified in the argument list forming a new object.
CARITHMETIC	Perform an arithmetic operation on the object and a constant replacing the object with the result.
CARITHMETIC3	Perform an arithmetic operation on a constant and an argument object, forming a new object.
NEW	Create a new permanent or temporary NDF object.
READ	Create an object with read acess by reading an NDF (or other format) file.
UPDATE	Create an object with update access from an NDF (or other format) file.
TEMPLATE	Create a new object based on a template NDF (or other format) file.
STATISTICS	Calculate various statistics on the data of an object.
SETCLASS	Set the class of an object.
GETCLASS	Get the class of an object.
SETRED	Mark the object as "reduced" by creating a reduced extension with value TRUE.
GETRED	Get the "reduced" status of an object.

FITSLIST	List the FITS header of an object (using MSG_OUT).
FITSITEM	List one "card" from the FITS header of an object.
HISTLIST	List the "history" extension of the object.

C.2 Class 1D

Source file: class_1d.f

Base Class: NDF

Class for objects with 1 dimensional data arrays.

Methods:

PLOT	Plot data for the object.

C.3 Class 2D

Source file: class_2d.f

Base Class: NDF

Class for objects with 2 dimensional data arrays.

Methods:

PLOT	Plot data for the object.
LOWPFILTER	Perform median filtering of the data.
AXFLIP	Flip axes of a 2D array.

C.4 Class GROUP

Source file: class_group.f

Base Class: NDF

Class for group objects.

Methods:

VALIDATE	Validate an object as a member of the GROUP class.
CREATE_GROUP	Create a new empty group object.
INSERT_OBJECT	Insert an object into a group.
DELETE_ENTRY	Delete an entry from a group.
GET_ENTRY	Get an entry from a group (by name or index).

GET_NENTRIES	Get the number of objects in a group.
DO_FOR_EACH	Invoke a specified method for each object in a group.
COMBINE	Combine data arrays of group objects (by mean, median etc).
MAKE_GROUP	Make a group from a list of objects.

C.5 Other Classes

There are a few other classes in Clalib, for example the polarimetry classes but these are not used in 2dfdr.

COL	???
GENERIC	???
GRAPH	???
INT	???
POL2D	???
POL	???

D Tdfred Classes and Methods

D.1 Class BIAS

Source file: cla_bias.f

Base class: CCDIMAGE

Class for bias frames.

Methods:

REDUCE	Reduce a bias frame
GETCLASS	???

D.2 Class BIASCG

Source file: cla_biascg.f

Base class: CALGRP

Class for groups of reduced bias frames.

Methods:

SELECT	Select the best matching bias frame from the group

D.3 Class BIASGRP

Source file: cla_biasgrp.f

Base class: GROUP

Class for groups of raw bias frames.

Methods:

REDUCE	Reduce the bias group by reducing the indivdual bias frames and combining them.

D.4 Class BIASRED

Source file: cla_biasred.f

Base class: 2D

Class for reduced bias frames.

Methods: none

D.5 Class CALGRP

Source file: cla_calgrp.f

Base class:GROUP

Base class for groups of calibration frames.

Methods:

SELECT	Select the best matching frame from the group.
INSERT_OBJECT	Insert an object into the group using a name derived from the run number.
COMBINE	Combine frames in the group using MEAN as the operation if there is only one frame.

D.6 Class CCDIMAGE

Source file: cla_ccdimage.f

Base class:2D

Class for CCD image data.

Methods:

DEBIAS	Debias the CCD data.
FLAT	Flat field correct the data.
DARK	Dark subtract the data.
CLEAN	Clean cosmic rays.
REDUCE	Reduce the CCD data frame.

D.7 Class DARK

Source file: cla_dark.f

Base class:CCDIMAGE

Class for dark frames.

Methods:

GETCLASS	???
REDUCE	Reduce a dark frame

D.8 Class DARKCG

Source file: cla_darkcg.f

Base class: CALGRP

Class for groups of reduced dark frames.

Methods:

SELECT	Select the best matching dark frame from the group

D.9 Class DARKGRP

Source file: cla_darkgrp.f

Base class: GROUP

Class for groups of raw dark frames.

Methods:

REDUCE	Reduce the dark group by reducing the indivdual dark frames and combining them.

D.10 Class DARKRED

Source file: cla_darkred.f

Base class: 2D

Class for reduced dark frames.

Methods: none

D.11 Class FFLATCG

Source file: cla_fflatcg.f

Base class:CALGRP

Class for groups of reduced fibre flat field frames.

Methods:

SELECT	Select the best matching fibre flat frame from the group

D.12 Class LFLAT

Source file: cla_lflat.f

Base class:CCDIMAGE

Class for long slit flat field frames.

Methods:

REDUCE	Reduce a long slit flat field frame.
HPFILT	Create a normalised high pass filtered flat field.

D.13 Class LFLATCG

Source file: cla_lflatcg.f

Base class:CALGRP

Class for groups of reduced long slit flat field frames.

Methods:

SELECT	Select the best matching long slit flat frame from the group.

D.14 Class LFLATGRP

Source file: cla_lflatgrp.f

Base class:GROUP

Class for groups of raw long slit flat field frames.

Methods:

REDUCE	Reduce the group by reducing the indivdual frames and combining them.

D.15 Class LFLATRED

Source file: cla_lflatred.f

Base class: 2D

Class for reduced long slit flat field frames.

Methods: none

D.16 Class MFARC

Source file: cla_mfarc.f

Base class:MFIMAGE

Class for multi fibre arc frames.

Methods:

REDUCE	Reduce the arc file. This simply turns off cosmic ray rejection and calls the reduce method of MFIMAGE.

FOCUS	Run a 2dF focus sequence on a pair of Hartmann shutter arc exposures.

D.17 Class MFFFF

Source file: cla_mffff.f

Base class: MFIMAGE

Class for multi-fibre flat field frames.

Methods:

REDUCE	Reduce the flat field file.
OVERTL	Plot a tram line map overlaid on the flat field image.

D.18 Class MFFLX

Source file: cla_mfflx.f

Base class: ???

Class for ???.

Methods: none

D.19 Class MFIMAGE

Source file: cla_mfimage.f

Base class: CCDIMAGE

Class for multi fibre image frames. This is the base class for all the individual types of multi-fibre frames and contains the methods associated with fibre extraction.

Methods:

EXTRACT	Extract the fibre data from a 2D data frame.
GFIBPOS	Determine fibre positions (the 2dfdr ``Find Fibres" command).
OVERTL	Plot a tram line map overlaid on the image.
GTLMAP	Generate a tram line map.
REDUCE	Reduce a multi-fibre image.
MATCH	Compare the multi-fibre image with a tram line map and determine the shift and rotation needed to align them.
FIBRELIST	List the fibres extension of an object.

FIBRE_INFO	List information on one fibre from the fibres extension.
LDSS_INFO	Add a fibre header for an LDSS image.
CRPROFILE	???
GETOFFSET	???
GREFPROF	???
GUESSUSED	???
GETCLASS	???

D.20 Class MFOBJECT

Source file: cla_mfobject.f

Base class: ???

Class for ???.

Methods: none

D.21 Class MFSARC

Source file: cla_mfsarc.f

Base class: MFSPEC

Class for extracted multi-fibre arc spectra.

Methods:

SCRUNCH	Scrunch (rebin onto a linear wavelength scale) an arc spectrum.
REDUCE	Reduce a set of multi-fibre arc spectra.
FOCUS	Reduce a pair of Hartmann shutter arc exposures to provide focus information for 2dF.

D.22 Class MFSFFF

Source file: cla_mfsfff.f

Base class: MFSPEC

Class for extracted multi-fibre flat field spectra.

Methods:

REDUCE	Reduce a set of multi-fibre flat field spectra.

D.23 Class MFSFLX

Source file: cla_mfsflx.f

Base class: ???

Class for ???.

Methods: none

D.24 Class MFSGRP

Source file: cla_mfsgrp.f

Base class: GROUP

Class for groups of reduced multi-fibre spectra.

Methods:

COMBINE	Combine a number of multi-fibre spectra.

D.25 Class MFSKY

Source file: cla_mksky.f

Base class: MFIMAGE

Class for multi-fibre offset sky frames.

Methods: none

D.26 Class MFSOBJECT

Source file: cla_mfsobject.f

Base class:MFSPEC

Class for extracted multi-fibre object spectra.

Methods:

PLOT	Plot reduced spectra.
REDUCE	Reduce multi-fibre object frame.

D.27 Class MFSPEC

Source file: cla_mfspec.f

Base class:2D

Base class for multi-fibre extracted spectra. This is the base class of the classes for individual types of multi-fibre spectra frames.

Methods:

SCRUNCH	Scrunch (rebin to a linear wavelength scale) the spectra.
PLOT	Plot the spectra.
FTPCAL	Correct spectra for fibre throughput.
FLATFIELD	Correct spectra using fibre flat field.
GFTPC	Get fibre throughput calibration.
SKYSUB	Do sky subtraction.
FIBRELIST	List fibre header information.
FIBRE_INFO	List header information on a specific fibre.

D.28 Class MFSPIRAL

Source file: cla_mfspiral.f

Base class:MFIMAGE

Class for multi-fibre data frames from a SPIRAL type instrument.

Methods: none

D.29 Class MFSSKY

Source file: cla_mfssky.f

Base class:MFSPEC

Class for extracted multi-fibre sky spectra.

Methods:

REDUCE	Reduce multi-fibre sky spectra.

D.30 Class MFSSPIRAL

Source file: cla_mfsspiral.f

Base class:MFSOBJECT

Class for extracted multi-fibre SPIRAL spectra.

Methods:

PLOT	Plot SPIRAL (IFU) data.

D.31 Class THPTCG

Source file: cla_thtpcg.f

Base class:CALGRP

Class for groups of fibre throughput functions (in the form of reduced offset sky spectra).

Methods

SELECT	Select the best matching fibre throughput function from the group.

D.32 Class TLMAP

Source file: cla_tlmap.f

Base class:2D

Class for tram-line maps.

Methods

SIMULATE	Generate a simulated data frame.

D.33 Class TMCG

Source file: cla_tmcg.f

Base class:CALGRP

Class for groups of tram-line maps.

Methods

SELECT	Select the best matching tram-line map from the group.

D.34 Class WAVECG

Source file: cla_wavecg.f

Base class:CALGRP

Class for groups of wavelength calibration data (i.e. reduced arc files).

Methods

SELECT	Select the best matching wavelength calibration file from the group.