EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

Data Management Division

Astronomical Catalog Library

User Manual

Doc.No. GEN-SPE-ESO-19400-0949

Issue 3.1

Date 16/01/98

Prepared	A. Brighton	16/01/98	
	Name	Date	Signature
Approved	M. Albrecht		
	Name	Date	Signature
Released	P. Quinn		
	Name	Date	Signature

Change Record

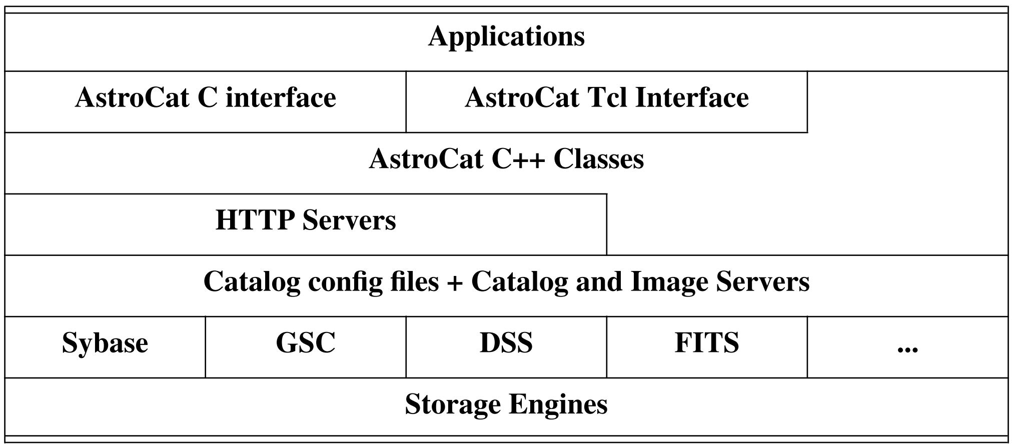
Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
2.7	24/03/97	All	Created from Design Specification document.
2.9	29/04/97	All	Minor corrections.
3.0	04/11/97	All	Added catalog directories, new config file keywords, new section describing config file.
3.1	01/16/98	All	Minor changes after split into *astrotcl* and *tclutil* packages, reference update.

1 INTRODUCTION

This manual describes a library for accessing astronomical catalogs and image servers locally and over the network in a generic way. The basic library consists of a set of C++ classes. In addition, C and Tcl interfaces are provided as well as Itcl widgets for browsing catalogs and catalog hierarchies. There is a separate library module for each of the three interfaces (C++, C and Tcl) and these will be described in more detail later in this document. The *cat* package described here is organized in the form of a Tcl package that can be loaded dynamically in a Tcl application or linked in as a library.

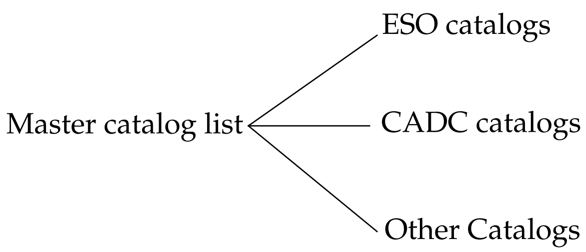
The main purpose of this library is to provide generic access to a variety of astronomical catalogs available locally or on the network. The interface hides the details of the various catalogs and allows you to access their contents in a uniform way.

Implementation Levels of AstroCatalog Libraries



The interface libraries provide methods to open and close a catalog, send a query or request and retrieve the results. In addition, methods are available to get information about the catalog contents, such as column names. The C and Tcl libraries are based on the C++ version.

Catalog configuration files contain the information needed to access the catalog servers. These files are specified as URLs (http:/... and file:/...) and so may be local files or automatically downloaded from anywhere on the network. The config files are simple ASCII files that contain *keyword: value* pairs describing each catalog. There are different types of catalogs, such as *catalog*, *archive*, *imagesvr*, *local* and *directory*. The last one is a special type of catalog that points to another catalog configuration file somewhere on the network. In this way, the lists of available catalogs form a tree of catalog severs.



1.1 Purpose

The purpose of this manual is to describe the *Astronomical Catalog* library interfaces and their implementation.

1.2 Scope

This document is primarily aimed at software developers using the catalog library package. Some sections are also aimed at those who would like to make catalogs available by developing HTTP catalog servers and making entries in a catalog configuration file. The package is also designed to be of use to ESO VLT applications (via a small layer that adds VLT standard CCS error logging) and to ESO Archive applications. This library is also currently used by Skycat, an astronomical image viewer based on the ESO RTD (Real-Time Display) application. Skycat uses the Tcl interface to provide support for navigating the catalog hierarchy, searching and editing catalogs, and plotting objects at the correct positions in images.

1.3 Applicable Documents

This document is based on the following documents:

[1] VLT-ICD-ESO-17230-0641, 1.0, 08/12/94 -- INTERFACE CONTROL DOCUMENT between Telescope Control Software and "Star Catalog System"

[2] VLT-SPE-ESO-17230-0640, 1.0 06/12/94 -- VLT Software - Requirements on Star Catalog System

[3] VLT-PRO-ESO-10000-0228, 1.0 10/03/93 -- VLT Software Programming Standards

1.4 Reference Documents

The following documents are referenced in this document:

[1] VLT-MAN-ESO-19400-1550 1.0 19/01/98 -- Tcl and C++ Utilities, Programmer's Manual

[2] VLT-MAN-ESO-19400-1551 1.0 19/01/98 -- Astronomical Tcl and C++ Utilities

2 OVERVIEW

The *Astronomical Catalog* library supports two basic types of catalogs. The first is a *catalog* containing data on astronomical objects, such as position, name, magnitude and so in. The second is an *image server* or a service that returns an image of an astronomical object, given its name or coordinates and a width and height. A search in a catalog returns a list of objects, while a request from an image server simply returns a (possibly compressed) FITS image file.

2.1 Catalog Configuration Information

The list of available catalogs and image servers is stored in one or more catalog configuration files, locally or somewhere on the net. An environment variable CATLIB_CONFIG may be set to the URL for the config file, otherwise a default URL is used that points to the central ESO catalog list. These config files are initially created by hand, usually by the same people who provide the catalog servers. Some applications, such as Skycat, provide interfaces for editing the config files and storing local copies of them. In this way users can keep a list of favorite catalogs in their own personal catalog config file. A detailed description of the config file format can be found at the end of this section.

All of the information in the catalog config files is available via the C++ and Tcl library interfaces, so for example, an application can easily determine whether or not a catalog supports World Coordinates and if so, which columns are used for RA and DEC.

2.2 Catalogs

Each type of astronomical catalog may contain a different set of columns. At least one of the columns should contain a unique id that can be used to identify the object. Catalogs that support world coordinates should have the columns *RA* and *DEC* in decimal degrees or *H:M:S D:M:S*.

Special Catalog Columns

Id	unique object Id
RA **or X**	World Coordinates: right ascension: α or X pixel position
DEC **or Y**	World Coordinates: declination: δ or Y pixel position
More	an HTTP URL pointing to more info on the object, such as can be displayed in a web browser (netscape)
Preview	an HTTP URL pointing to an image of the object (DSS or HST, for example)

By default, the first 3 columns are assumed to be *ID*, *RA* and *DEC.* The column positions can be overridden in the catalog configuration file.

Not all catalogs support World Coordinates. Some may have the position specified in image pixel coordinates. These catalogs are also supported, but require a special entry in the catalog configuration file indicating the positions of the X and Y columns.

A catalog is not required to have any position columns. If the catalog contains neither world nor image coordinates, the catalog configuration file entry should indicate this by specifying a value of -1 for the column indixes of ra, dec, x and y.

There may be any number of other columns. Certain column names, such as **More** and **Preview**, are taken to have special meanings by some applications (such as Skycat).

One of the most common operations on catalogs is a circular search, which returns a list of objects within a given radius range of a center position.

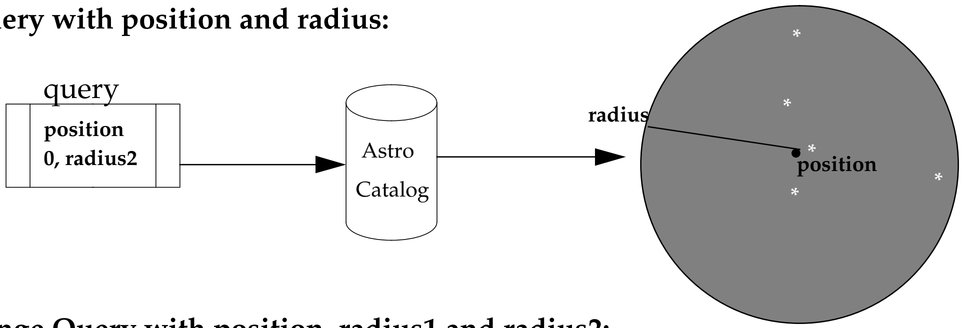
Circular Search Attributes

Position	Position (α, δ), (x, y) or name (using a name server, such as SIMBAD or NED to convert name to a WCS position).
Radius1, Radius2	Minimum and Maximum radius from center position (α, δ) in arcmin or from image pixel (x, y) in pixels.

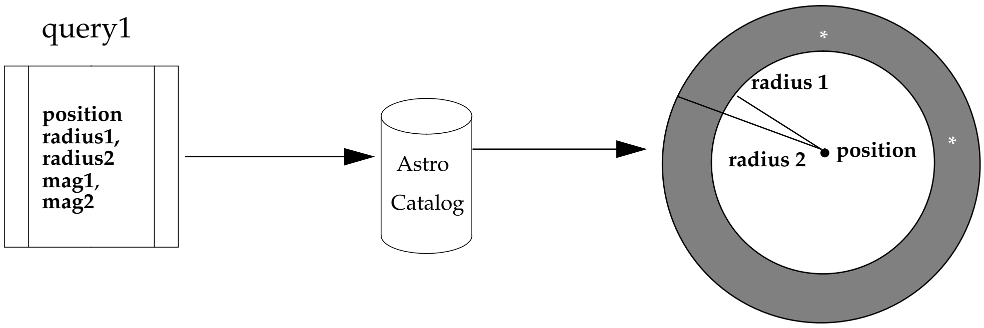
In addition to the above attributes, that apply to any catalog having position coordinates, it may be possible to search for a range of values for any of the other catalog columns. This depends on the way the catalog server is implemented. An entry in the catalog configuration file specifies a list of *search columns* for each catalog.

A simple catalog query might specify a name or world coordinates position and a radius and get as a result a list of all of the objects in the given circle. A more complicated query would specify a minimum and maximum radius, and perhaps a minimum and maximum magnitude.

1. Query with position and radius:



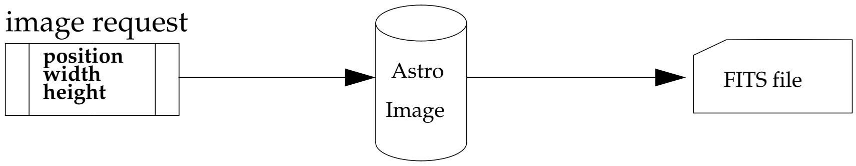
2. Range Query with position, radius1 and radius2:



In each query, only the **non-null** fields are considered. A special *null position* is defined by library interface. Note that not all types of queries are supported by all catalog servers.

2.3 Image Servers

To retrieve an image from an image catalog or server, you need to specify the coordinates (either as a name or world coordinates position) and the desired width and height of the image. Since the return value from an image server request is a FITS format image, it is easiest to return the name of a temporary file containing the image.



The relevant image information can then be taken from the FITS header and the image can be displayed with an image viewer.

2.4 Catalog Servers Implementation

Remote catalogs (located on a remote host in the network) are accessed via HTTP servers. For each catalog, a simple wrapper application is developed, to be used as an HTTP *cgi-bin* application. That is, a catalog server is accessed via a standard URL that executes an application with the given arguments and returns the results. These cgi-bin applications are typically perl scripts or simple C programs that access a database.

Local catalogs are implemented as simple ASCII files in *tab table* format. This is the same format that is used to return query results from remote catalogs. There is also a suite of tools available in the public domain for manipulating tab tables called *Starbase*. The Astrocat library currently uses some C++ classes (`TabTable, QueryResult, TcsQueryResult`) to implement local catalogs.

The results of catalog queries are always returned in the format of a *tab separated table* of values. The column headings are separated from the data by a *dashed* line:

```
Title
# comments
Keyword: value

...
ID            RA            DEC           MORE          PREVIEW
--            --            ---           ----          -------
NGC5457-FD2   210.593704    54.299526     M=http://...  P=http://...
NGC5457-FLD2  210.593704    54.299526     M=http://...  P=http://...
...           ....          ...           ...           ...
```

The catalog servers hide some of the details of the individual catalogs by following certain conventions. For example, by default the first three columns are ID, RA and DEC (in degrees) and the equinox is J2000. These defaults can be changed in the catalog configuration file or by inserting catalog configuration information in the header of the query results in the same format as in the config file.

The MORE and PREVIEW columns are optional, but are assumed to contain valid URLs if present, pointing to more information (for MORE) or a FITS image (for PREVIEW). In addition there may be any number of other columns.

The catalog library gets the list of catalog and image servers from a configuration file, which may be a local file or a remote file retrieved via HTTP. The config file lists, for each server, the URL to use to access the catalog. Substitution is performed on the URL to insert the query arguments, such as *id*, *ra*, *dec*, *radius range*, *mag range*, etc.

For remote catalogs, the catalog library interface sends the query URL to the HTTP server on the given host and parses the result. Queries in local catalogs are handled internally. Once a query has been made, an application can access the column headings and values via the library interface. More details on this will be presented later in this document.

2.5 Adding New Catalogs

Local catalogs can be accessed directly on the local system. They may have - but do not require - an entry in a catalog config file, since you can simply specify the file path name as the name of the catalog.

If the URL given for a catalog in the config file is the name of a local command, the command will be executed and should write query results in tab table format, as described above, on the stdout. Before executing the command, variables, such %ra, %dec or %x, %y (see below) are substituted by the catalog library to form the command string.

The first thing to do to make a catalog available over the net via the catalog library is to write an HTTP cgi-bin application to access the catalog. This application receives it's arguments from the environment variable QUERY_STRING, forms the query to send to the underlying database and then formats the results in tab table format before writing them on the stdout. The catalog library performs variable substitution (%ra, %dec, etc., see below) on the URL before sending it to the server.

Besides the column headings and the data, the query result may contain the catalog config entry, or parts of it, to indicate, for example, the locations of the RA and DEC columns or the symbols to plot.

2.6 The Catalog Configuration File

A catalog configuration file contains the information necessary to access catalogs. The syntax of the file is as follows:

- Blank lines and lines beginning with # are ignored.
- Lines ending with a backslash `\` are continued on the next line.
- A catalog description entry is made up of *keyword: value* pairs, one per line.
- The first keyword in every catalog entry is `serv_type`, which gives the type of catalog.
- The order of other keywords is unimportant (many keywords are optional).
- Any unknown keywords are ignored (*for compatibility with future extensions*).
- Any catalog entry with an unknown serv_type is also ignored (*for compatibility*).
- Many of the keywords represent query options (*sort columns, order, etc.*) that may not be supported by all catalog servers.
- For all keywords requiring a list of values, the basic format is that of a Tcl list. Normally this is simply a list of words or strings in double-quotes separated by white space. However curly braces `{` and `}` may be used in place of quotes.
- If a keyword's value may contain more than one list, the lists are separated by a colon ":".
- If a keyword's value may contain a variable reference (*see* **symbol** *below*), the variable is expected in Tcl format. For example: `$ra`, `${a/b}`, `${x pos}`.

The syntax for each catalog entry is:

Keyword	Value
serv_type	Service type, one of: `catalog`, `archive`, `local`, `namesvr`, `imagesvr`, `directory` (see *Service Types* below)
long_name	Long name of catalog, suitable for displaying in a user interface.
short_name	Short name of catalog.
url	URL used to access the catalog. Variables such as %ra,%dec, etc. are expanded (see *Variable Substitution in URLs* below).
symbol	List indicating how to plot symbols for given column values (see *Plot Symbols* below).

Keyword	Value
copyright	Optional copyright notice for catalog to display in user interface.
search_cols	Optional list of column names and min and max value labels for columns that can be used to search the catalog, in the format: `col1 minLabel maxLabel : col2 minLabel maxLabel : ...` Example: `search_cols: mag "Brightest (min)" "Faintest (max)"` The min and max labels are to be displayed in a user interface when entering the condition arguments for a catalog query.
sort_cols	Optional list of columns to sort by in the format: col1 col2 col3 ...
sort_order	Optional sort order: Set to `increasing` or `decreasing`.
show_cols	Optional list of columns to display (default: all). Besides indicating which columns to display or hide after a query, this list also indicates the order in which the user interface should present the columns. This keyword may be implemented in the user interface or optionally also in a catalog server, if it supports the option.
id_col	Column containing a unique id field (default: column 0).
ra_col	Column containing RA position (for catalogs supporting WCS, default: column 1)
dec_col	Column containing DEC position (for catalogs supporting WCS, default: column 2)
x_col	Column containing X image pixel coordinate (for catalogs not supporting WCS)
y_col	Column containing Y image pixel coordinate (for catalogs not supporting WCS)
is_tcs	The optional value may be set to 1 to indicate that the catalog is in or should be converted to TCS format. TCS catalogs have fixed column names and rules for conversion of column names from other catalogs.

Note: Column indexes, such as *ra_col*, *dec_col*, *x_col*, and *y_col* may be set to -1 to indicate that there is no such column.

2.6.1 Catalog Service Types

The currently supported catalog service types (value for **serv_type** keyword) are:

Service Type	Description
catalog	Server returns a tab separated table of row/col values.
archive	Is similar to catalog, but is used for catalogs representing archives of images.
namesvr	Server returns a single line with id, ra and dec to resolve the given object name.

Service Type	Description
imagesvr	Server returns an image file in FITS format. The image may be compressed, however the **Content-type** should indicate the type of compression: • image/x-hfits means the image is H compressed. • image/x-gfits or image/x-gstarbase means image is gzipped. • image/x-cfits or image/x-cstarbase means UNIX compressed. • text/html is taken to be an error message in HTML format.
local	A local catalog file in tab table format (same format as the output of the catalog servers).
directory	This is a special entry where the URL is a pointer to another catalog config file, used to build up a tree of catalog providers.

2.6.2 Variable Substitution in URLs

The **url** keyword value is used to build a URL to send to a catalog server or execute as a local command. The syntax for HTTP based catalog servers is as follows:

```
url: http://host:port/cgi-bin/server?arg1&arg2&...argn
```

If *:port* is missing, it defaults to 80. If the URL refers to a local command, the syntax is like this:

```
url: /pathname/command arg1 arg2 ... argn
```

Substitutions are performed on the URL as follows, based on the query arguments set in the application:

Variable	Description
%ra, %dec	World coordinates of center point (for catalogs supporting World Coordinates).
%x, %y	Image pixel coordinates of center point (for pixel based catalogs).
%w, %h	Width and height in arcmin (if used with %ra, %dec) or image pixels (for %x, %y).
%r1, %r2	Min and max radius (for circular query) in arcmin (for %ra, %dec) or pixels (%x, %y). The order is unimportant.
%r	Use when server only accepts single radius value.
%m1, %m2	Min and max magnitude (order unimportant, see also %cond for an alternative)
%m	Use when server only accepts single magnitude.
%n	Max number of rows to return.
%cols	List of columns to return. Default (empty) means return all columns. The column names, if any were specified, are inserted in the format: *col1,col2,...coln*.
%id	ID field of item to return, for search by unique object id.
%mime-type	Value for HTTP mime-type field to insert in HTTP get command.
%sort	Inserts a list of sort columns in the format: *col1,col2,...,coln*. An empty list means none or default sorting.

Variable	Description
%sortorder	Inserts the string: `increasing` or `decreasing` to indicate the sort order.
%cond	Inserts a search condition string, if any was specified in the query, in the format `colName1=minVal,maxVal&colName2=minVal,maxVal,..` The catalog server should return only rows where the given columns have values in the given ranges.

2.6.3 Plot Symbols

The **symbol** keyword value describes how to plot and label objects in a user interface based on the values in a query result row. You can, for example, specify that you want yellow circles drawn around guide stars from the GSC catalog with a size based on the magnitude of the object in image pixels. You can also specify the shape and orientation of an ellipse to be drawn around each object, based on the values in each query result row and label each object with a variable string. The syntax for the **symbol** keyword value is as follows:

symbol: *colNames symbolInfo sizeExpr : colNames symbolInfo sizeExpr : ...*

where:

colNames is a list of column names that may be used as variables in *symbolInfo* or *sizeExpr*. Only these column names will be available as variables. The names should be separated by spaces. Column names that contain embedded spaces or special chars (*Tcl syntax*) should be placed in "*double-quotes*" or {*braces*}.

symbolInfo is the name of a symbol to plot, or optionally a list of the form:

symbol color ratio angle label condition

Only *symbol* is required, the other values are all optional.

- *symbol* is the symbol to plot and may be one of: `square`, `circle`, `triangle`, `cross`, `diamond`, plus, `ellipse`, `compass`, `line`, `arrow`. Some of these symbols (the last 5 in the list above) are typically plotted in the orientation of the image, i.e.: they are rotated relative to the world coordinate north of the image, if it is known (see also *angle* below).
- *color* is the color of the symbol and may be any valid X color name, such as `red`, `yellow`, `blue`, etc. The default is to draw the symbol in black and white, so that it will always be visible in any color image.
- *ratio* and *angle* are only used for symbols where it makes sense, such as `ellipse`, `line`, `arrow`, `compass` and `plus`. The *ratio* gives the shape of the symbol as a ratio *width/height* and *angle* is the rotation angle of the symbol, relative to WCS north. Both values may be expressions using the column names given in *colNames* above.
- *label* may also be an expression or text mixed with an expression and is used to label each object.
- *condition* may be an expression (using the column names as variables) that evaluates to a boolean value (1 for *true*, or 0 for *false*). If the condition evaluates to 1, the symbol is displayed for that object, otherwise it is not. Multiple symbols may be used, each with a different condition, so that different symbols, shapes and colors are used depending on the column values.

sizeExpr is an expression in terms of the column names given by *colNames* above and is used to determine the size of the symbol. It may also be a list of *{expr units}*, where:

- *expr* gives the size of the object (a valid Tcl expression using column names as variables).
- *units* is either `image` (*for image pixels)*, "`deg J2000`" or "`deg B1950`" (*for World Coordinate degrees in the given equinox*).

examples:

symbol: mag circle 15-$mag
Plot circles with the size 15-$mag image pixels.

symbol: mag {circle yellow} {$mag/60 {deg J2000}} **:** rmag {square red} {$rmag*3.5}
Plot yellow circles with the size $mag/60 degrees (J2000) and red squares with the size $rmag*3.5 (image pixels).

symbol: {a/b pa mag} {ellipse yellow ${a/b} $pa mag=$mag} {15-$mag}
Plot yellow ellipses with the ratio ${a/b}, the angle $pa, and the size 15-$mag (image pixels) and label each object with the text "mag=$mag" (inserting the value for $mag).

symbol: "a/b pa mag" "ellipse yellow ${a/b} $pa mag=$mag" "15-$mag"
Same as above, but using "double-quotes" instead of {braces}.

symbol: "a/b pa mag" "ellipse yellow ${a/b} $pa mag=$mag $mag<12" "15-$mag" \
: "a/b pa mag" "cross blue ${a/b} $pa mag=$mag $mag>=12" "15-$mag"
Same as above, but plot yellow ellipses for symbols with mag less than 12 and blue crosses for the others..

Remember that whenever there are ambiguities (embedded spaces or special chars), the Tcl list and variable syntax is used. This makes it very easy for Tcl applications to evaluate the expressions.

If you are not able to calculate the correct size of an object in an expression, there might still be a way to do what you want (at least for your own catalogs). You can insert a call to a Tcl proc or command in the expression. For example:

symbol: "rmag bmag" circle "[my_plot_proc $rmag $bmag]"
Plot circles with a size in image pixels calculated by the Tcl proc my_plot_proc.

Note that the cat package does not actually plot any symbols or deal with images directly. See the *skycat* package for that. It defines subclasses of some catalog user interface classes and adds image plotting support.

2.6.4 Example Catalog Config Entry

Here is an example catalog config entry for a catalog implemented by a local command:

serv_type:	catalog
long_name:	My test catalog
short_name	test1@local
url:	/usr/local/bin/test1.cat \ -pos "%ra %dec" \ -radius "%r1 %r2 " \ -cond "%cond" \ -sort "%sort" \ -sortorder %sortorder

symbol: “mag pa a/b” \
“ellipse red ${a/b} $pa mag=$mag” \
15-$mag

search_cols: mag “Brightest (min)” “Faintest (max)” \
: pos-e “Min Pos Err” “Max Pos Err” \
: ppa “Min ppa” “Max ppa”

sort_cols: mag ppa

sort_order: decreasing

id_col: 2

ra_col: 3

dec_col: 4

This entry defines a catalog with the name “*My test catalog*” (alias *test1@local*) that is implemented by a local command (script or application) */usr/local/bin/test1.cat*. A user interface, such as the catalog window in *Skycat*, will display entries to enter min and max values for *mag*, *pos-e* and *ppa*. The entries will be labeled **Brightest (min):**, **Faintest (max):**, and so on.

When a query is made, the command (test1.cat) will be called with the argument variables expanded. For example:

```
/usr/local/bin/test1.cat \
    -pos “03:19:48.24 +41:30:40.31” \
    -radius “0 10“ \
    -cond “mag=0,15&pos-e=0,1&ppa=45,90” \
    -sort “mag,ppa” \
    -sortorder decreasing
```

The command should output a query result in tab table format. In this case the column indexes for the *id*, *ra* and *dec* fields are 2, 3 and 4, resp. An image display application such as *Skycat* will then plot each row of the result as a red ellipse with the ratio ${a/b}, and the angle $pa and label each object with the label “mag=$mag”.

3 USER'S GUIDE

This section gives an overview of the *Astronomical Catalog Library* interface. For details, see the man pages in the reference section.

The basic library interface is implemented in C++ and consists of a collection of classes for dealing with astronomical catalogs and image servers. C and Tcl interfaces are also provided for convenience as well as Itcl user interface classes for browsing catalogs and catalog hierarchies. Each interface is described separately below.

3.1 C++ Interface Library

3.1.1 World Coordinates in C++

Arguments representing world coordinate positions are required in all three interfaces. There is a central C++ class called `WorldCoords` that handles the conversion between the various formats (H:M:S [+-]D:M:S, floating point degrees, char string,...) and equinox (1950, 2000, ...). If a user enters an astronomical object name in a user interface, it can first be resolved via a name server, such as SIMBAD or NED (using the Catalog library routines) to get the world coordinates. The default equinox is J2000, however any floating point number may be specified as the equinox when working with world coordinates. The world coordinate utilities support equinox conversion.

The interfaces for dealing with world coordinates are defined in the *astrotcl* package, which is a collection of astronomical software utilities. See the *astrotcl* documentation for details.

3.1.2 Catalog Classes

The C++ interface library consists of a collection of classes that interface to the catalog and image servers or manipulate the local catalog files. There are two main classes: *AstroCatalog*, for information catalogs and *AstroImage* for image servers. There is also the *AstroQuery* class, representing queries and the *QueryResult* class for representing query results. Local catalogs are implemented by a subclass of *AstroCatalog* called *LocalCatalog*. There is also a set of classes for dealing with special TCS (Telescope Control Software) catalogs (*TcsCatalog*, *TcsCatalogObject*, *TcsQueryResult*). The following table lists the main C++ classes and their uses:

Catalog C++ Classes

Class	Description
AstroCatalog	Implements basic remote HTTP catalog access.
AstroImage	Used to access remote image servers to fetch images given coordinates.
LocalCatalog	Subclass of AstroCatalog, implements local catalogs (see also QueryResult).
TcsCatalog	Subclass of AstroCatalog, used for TCS catalogs, which have fixed columns.
TabTable	This generic class is used to manage tab tables in memory.
QueryResult	Subclass of TabTable, used to represent the results of a catalog query or a list of rows to be inserted in or deleted from a local catalog.

Catalog C++ Classes

Class	Description
TcsQueryResult	Subclass of QueryResult specialized in TCS style catalogs.
AstroQuery	Object used to represent a query (position, radius, magnitude range, etc.).
CatalogInfo	Class representing the contents of the catalog configuration files, contains a hierarchical list of catalog entries.
CatalogInfoEntry	This class represents an entry in a catalog configuration file.
TcsCatalogObject	Represents one row or object in a TCS style catalog.
TclAstroCat	Implements the `astrocat` Tcl command (Tcl interface to catalogs).
TclAstroImage	Implements the `astroimage` Tcl command (Tcl interface to image servers).
TclTcsCat	Tcl access to TCS catalogs.

See the man pages in the reference section for details on the classes listed above.

3.1.3 Determining Which Catalogs are Available

In order to access a catalog, you need to know its name. This will most likely be selected from a menu of available catalogs in a user interface. This section describes how to find out which catalogs are available and from where.

3.1.3.1 Navigating the catalog lists

A list of available catalogs can be accessed via static member functions in class CatalogInfo, the class responsible for reading and parsing the catalog configuration files. The following code demonstrates how you can access the default list of catalogs:

```
#include "CatalogInfo.h"
...
for(const CatalogInfoEntry* e = CatalogInfo::first(); e != NULL;
      e = e->next()) {
      cout << e->servType() << ": " << e->longName() << endl;
}
```

The above example prints out the service type and long name of all of the catalogs in the default catalog list.

3.1.3.2 Hierarchical catalog lists

The catalog list is hierarchical in that it may contain catalog *directory* entries that point to other lists of catalogs. The `next`() method returns the next entry in the list, while the `link`() method returns a pointer to the first entry in a catalog directory, or NULL if the link does not exist. The following function traverses the list of known catalogs and returns a pointer to the named catalog entry or NULL if it was not found (taken from the *CatalogInfo::lookup* method):

```
CatalogInfoEntry* lookup(CatalogInfoEntry* entry, const char* name) {

      for (CatalogInfoEntry* e = entry; e != NULL; e = e->next())
            if (strcmp(e->longName(), name) == 0
             || strcmp(e->shortName(), name) == 0)
                  return e;

      for (CatalogInfoEntry* e = entry; e != NULL; e = e->next()) {
            if (e->link()) {
                  CatalogInfoEntry* p = lookup(e->link(), name);
                  if (p)
                        return p;
            }
      }
      return NULL;
}
```

Note that directory links are not followed automatically, since this could involve network delays or even endless loops. By default the `link()` method returns NULL unless the config file that the link points to is explicitly loaded like this:

```
if (CatalogInfo::load(dirEntry) != 0)
      return ERROR;
```

The `CatalogInfo::load` method takes a `CatalogInfoEntry` pointer for a catalog directory, follows the URL, loads the new config file and sets the `link()` member of `dirEntry` to point to the first entry in the new list.

3.1.3.3 Catalog Entries

Catalog entries exists for each catalog and are represented by objects of class CatalogInfoEntry. This class has methods for all of the known configuration file keywords. The methods all have 2 versions. With no arguments, they return the value, and with one argument set the value.

The `servType()` method returns the type of server. There are currently these types supported:

Catalog Server Types

Server Type	Description
catalog	A standard astronomical catalog that returns a tab separated table of query results, with column headings separated by a dashed line from the data.
archive	Same as *catalog*, but used to distinguishes *catalogs* from *archives* in user interfaces.
local	A local catalog, implemented as a tab table - an ASCII file with tab separated columns.
namesvr	This is a special type of catalog that is used to resolve astronomical object names into world coordinates. Currently SIMBAD and NED are used as name servers.
imagesvr	This type of server returns a (possibly compressed) FITS image as a result.
directory	With this entry, the URL points to another config file. When the file is loaded, the `link()` method points to the first entry in the new list.

The other methods in the `CatalogInfoEntry` class all correspond to fields in the catalog config file. These fields were described in detail earlier in this document. Currently the following methods are available, each with versions for setting and getting the values:

Catalog Config Entry Methods

Method	Description
servType	Get or set the catalog service type
longName	Get or set the long name of catalog
shortName	Get or set the short name of catalog.
url	Get or set the URL used to access the catalog.
symbol	Get or set the list indicating how to plot symbols.
copyright	Get or set the copyright notice for catalog to display in user interface.
searchCols	Get or set the list of column names and min and max value labels for columns that can be used to search the catalog.
sortCols	Get or set the list of columns to sort by.
sortOrder	Get or set the sort order: Set to `increasing` or `decreasing`.
showCols	Get or set the list of columns to display and the order of display.
id_col	Get or set the column index for columns containing the id field.
ra_col	Get or set the column containing the RA position.
dec_col	Get or set the column containing DEC position.
x_col	Get or set the column containing the X image pixel coordinate.
y_col	Get or set the column containing the Y image pixel coordinate.
is_tcs	Get or set the value of the tcs flag.

The default catalog configuration file is automatically read the first time it is needed. There is a default URL that points to the standard configuration file on archive.eso.org. You can override the default to add your own catalogs by defining the environment variable CATLIB_CONFIG[1] to a valid URL. The URL can point to a remote HTTP server ("http://host/dir/configfile") or it can point to a local file ("file://dir/configfile"). If the config file is not found, a hard coded default is used.

3.1.4 Working with Catalogs in C++

An astronomical information catalog is opened with the static member function `open`:

```
AstroCatalog* cat = AstroCatalog::open(catalogName);
if (!cat) {
      error(...);
}
```

The catalog name can be either the *long name* or the *short name*, as defined in the catalog config file. The return value from the call to `open` is a pointer to an `AstroCatalog` object allocated for the

1. The environment variable SKYCAT_CONFIG is also accepted, for compatibility with previous versions.

given catalog. It should be deleted with the C++ *delete* operator when no longer needed.

If the argument passed to `AstroCatalog::open` is the name of a file, the return value will point to a subclass of `AstroCatalog` called `LocalCatalog`, which works with local catalog files, but otherwise has the same basic interface:

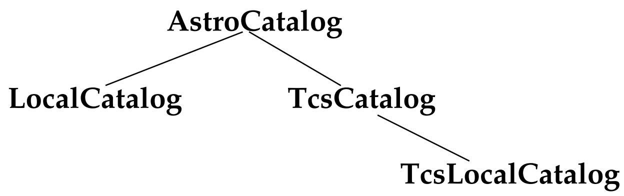
```
AstroCatalog* cat = AstroCatalog::open(fileName);
if (!cat) {
      error(...);
}
```

For TCS catalogs[1], the calling sequence is similar:

```
TcsCatalog* cat = TcsCatalog::open(catalogName);
```

or for local catalogs, if the argument is a file name, the return value points to a derived class of `TcsCatalog` called `TcsLocaCatalog`, which is specialized in local TCS catalog files.

The tree below illustrates the class hierarchy of the basic catalog access classes mentioned above:



3.1.4.1 Catalog Queries

There are a number of methods defined in the AstroCatalog classes for making queries. The main method is *AstroCatalog::query*. To use it, you first create an `AstroQuery` object to represent the query and pass it, along with some other arguments, to the *query* method:

```
WorldCoords pos(3, 19, 48, 41, 30, 39);
char* filename = NULL;
AstroQuery q;
q.pos(pos);
q.radius(0, 10);
q.maxRows(10);
AstroObject* result = NULL;
int numRows = cat->query(q, filename, result);
```

The `AstroQuery` object holds the query parameters. Any fields that are not specified are taken to be null and are not used in the query. In the example above, the `WorldCoords` class (see *WorldCoords(3)* in the *astrotcl* package) was used to create the center position argument as 3:19:48 +41:30:39 and a radius range was specified for the query. The table below lists all of the possible query parameters. In most cases, these are the names of two `AstroQuery` methods - one to set the value (with 1 argument) and another to get the value (with no arguments).

Class AstroQuery Public Methods/Attributes

Method	Description
id	Object Id from a previous query (if specified, other fields are ignored)

1. TCS catalogs are special catalogs with fixed columns, used by the ESO VLT Telescope Control Software.

Class AstroQuery Public Methods/Attributes

Method	Description
pos (α, δ, equinox) or (x, y)	Center position for query (class `WorldOrImageCoords`). The value represents either a World Coordinate position (*ra, dec, equinox*) or an image (pixel) coordinate position (*x, y*).
mag (mag1, mag2)	Min and max magnitude of object. If the catalog does not contain a column called "mag", these are ignored.
radius (radius1, radius2)	Find objects at a distance of between *radius1* and *radius2* from the center position, or up to a distance of *radius*. The units of the radius values depends on the position argument and are either *arcmin* for a World Coordinate position or *image pixels* for an image coordinate position.
maxRows	Maximum number of rows to return.
width, height	Dimensions of area to search (converted internally to a center position and radius). the units are the same as for radius above.
colNames, numCols	List of column names to return, the number of column names.
sortCols numSortCols sortOrder	List of column names to sort by, the number of sort columns, the sort order (*increasing* or *decreasing*).
condition searchCols numSearchCols minValues maxValues	These attributes are used together to specify a simple *condition* to the query. The condition is that the given columns (*searchCols*) have values between *minValues* and *maxValues*. All the values are specified in string form, inorder to be type independent, however the comparisons are still done numerically where applicable. Each of the lists have the same length: *numSearchCols*.

The actual query is made by the *`query`* method, which takes the *`AstroQuery`* object and an optional *filename*, does the query and returns the number of objects found. On success, the last parameter, *result*, is filled with the results of the query.

3.1.4.2 Query Results

The *QueryResult* class represents the result of a query and provides methods to get the column names and the value at a given row and column. As an example, the following code would print out the result of the query:

```
char* value;
for (i = 0; i < result.numRows(); i++)
    for (j = 0; i < result.numCols(); j++) {
        if (result->get(i, j, value) == 0)
            cout << "row: " << i << ", col: " << j << ": " << value << endl;
        else error(...);
    }
```

In this case, the column *`value`* was declared as string, however, there are also overloaded methods that extract result values as *ints*, *shorts*, *floats*, *doubles*, and also as world coordinates (class `World-Coords`) in a given format and equinox.

The following table lists some of the public methods in class `QueryResult` (and its base class

TabTable) for accessing the results of a query.

Class QueryResult Public Methods

Method	Description
numRows	Return the number of result rows.
numCols	Return the number of result columns.
colIndex	Return the result column index for the given result column name.
colNames	Return a pointer to an array of result column names.
get	Get result values by row and column index or name. Multiple overloaded versions of this method support specifying the column as an index or column name and getting the type as an `int`, `short`, `char`, `char*`, `float or double`.
getPos	Get the position value for the given result row.

A number of `QueryResult` methods were not listed here because they are only used to implement *local catalogs* and are not normally needed otherwise.

3.1.4.3 TCS Catalogs

For the *TcsCatalog* class, the columns of the result are fixed, so there is no need to access them as strings. In this case, the call to the *query* method should pass an object of type *TcsQueryResult* rather than its parent class *QueryResult*. The TcsQueryResult class extends the QueryResult class by adding a method to access rows as objects.

```
TcsQueryResult result;
int num_results = cat->query(q, NULL, result);
```

Then you can access the query result rows as objects, as in the following example:

```
TcsCatalogObject obj; // holds data for one row
for (int row = 0; row < num_results; row++) {
      if (result.getObj(row, obj) != 0)
            cout << "row " << row << ": ERROR\n";
      else
            cout << obj << endl;
}
```

For TCS catalogs, replace the *get(row, ...)* methods with the following:

Method	Description
getObj(row, obj)	Put the contents of the given query result row in the given object of type class TcsCatalogObject.

A Tcs catalog row always contains the following columns (any missing columns are set to a predefined null value):

TCS Catalog Columns

Column	Type	Description
id	char*	Object catalog id
ra	double	Alpha coordinate for the target in decimal degrees
dec	double	Delta coordinate for the target in decimal degrees
cooSystem	char*	Equinox system and equinox ("B1950" or "J2000")
epoch	double	Epoch expressed as decimal year
pma	double	Proper Motion Alpha in radians/year (-10.0 to 10.0)
pmd	double	Proper Motion Delta in radians/year (-10.0 to 10.0)
radvel	double	Radial velocity in km/sec (-200000 to 200000)
parallax	double	Parallax in arcseconds (-10000 to 10000)
cooType	char*	Coordinate type as "M" for mean or "A" for apparent character
band	char*	Magnitude wavelength band ("V")
mag	double	Object's magnitude in given band
more	char*	An HTTP URL pointing to more info on the object
preview	char*	An HTTP URL pointing to an image of the object
distance	double	Distance to center of the field
pa	double	Position angle based on center of the field

3.1.4.4 Other Query Methods

A number of other methods are less generic, but were defined as a convenience for common queries[1]. For example,

```
cat->GetDescription(numCols, colNames);
```

sets the parameters to the number of columns in the catalog and the column names. If you have an object Id, you can retrieve the object with the following call:

```
status = cat->getObject(id, numCols, colNames, result);
```

You pass in the object *id* and the names of the columns you want and get back an array of column values (as strings). To perform a rectangular *area* query, specifying two points, you can call the following method:

```
status = cat->getArea(numCols, colNames, pos1, pos2, mag1, mag2, maxRows,
                      filename, numFound, result);
```

If `filename` is specified, the result is also copied to the given file. In any case, you can access the

1. Actually these methods were required to implement the ESO VLT C interface specification for catalog access.

result via the `result` argument, which is filled with the data on return. A *circular* search, where you specify a center point and one or two radius values, can be made as follows:

```
status = cat->circularSearch(numCols, colNames, pos, radius1, radius2,
        mag1, mag2, maxRows, filename, numFound, result);
```

Again, the results are written to the given file, if specified and to the given result object. To get the closest star or object to a given world coordinate point, you can call this method:

```
status = cat->searchClosestStar(numCols, colNames, pos, mag1, mag2, result);
```

In this case, the result only represents a single object or row. The following table summarizes the public methods of the AstroCatalog class:

Class AstroCatalog Public Methods

Method	Description
open	Open the named catalog and return a pointer to an AstroCatalog object created for it, or NULL if errors occur.
query	Pass a query to the catalog and return a the number of objects found. Return result as a pointer to a *`QueryResult`* object.
numCols	Return the number of columns in the catalog.
colName	Return the name of the given column (0 is first col).
hasCol	Return true if the catalog has the named column.
colIndex	Return the index of the named column.
symbol	Return the symbol entry for the catalog, to use to plot the catalog objects.
nameToWorldCoords	Convert an astronomical object name to World Coordinates using the given name server (SIMBAD by default).
status	Return status (after constructor) for error checking.
name, longName, shortName	Return the name (longName) or shortName of this catalog.
more	Return true if more than *nrows* rows would have been available to the last call to query().
getDescription	Return the number of catalog columns andthe column names.
getObject	Return the values for a specific object, given by Id.
getArea	Return the values for objects in a given rectangular area, given by two points.
circularSearch	Return the values for objects in a circular area given by a point and one or two radius values.
searchClosestStar	Return the values for the object closest to the given point.
getError	Return the text of the most recent error message.
getErrorCode	Return the error code (errno value) fro the most recent error message.
clearError	Reset the saved error message to empty.
feedback	Arrange to have feedback information written to a given FILE while waiting for the query to complete (used to display in a user interface).

3.1.5 Image Servers in C++

The `AstroImage` class interface is similar to that of class `AstroCatalog described above`. Here we also have an `open` method to connect to the image server:

```
AstroImage* im = AstroImage::open(serverName);
if (!im)
      error(...);
```

The return value from `AstroImage::open` is a pointer to an `AstroImage` object created for the given image server. To retrieve an image from the image server, you need to specify a position (or object name) and a width and height for the image:

```
WorldCoords pos(3, 19, 48, 41, 30, 39);
double width = 2.0, height = 3.5;
if (im->getImage(pos, width, height) == 0)
      displayImage(im->tmpfile());
```

Here the position is given as 3:19:48 +41:30:39, and the returned image should be 2.0 x 3.5 arc minutes in size (centered at the position). The return value is the status (0 is OK, 1 for errors). The name of the temporary file containing the requested FITS image can be accessed via the AstroImage::tmpfile() method. The file should be copied if necessary, since it will be rewritten in the next call to `getImage`.

Class AstroImage Public Methods

Method	Description
open	Open the named image server and return a pointer to an AstroImage object created for it, or NULL if errors occur.
getImage	Pass a query to the catalog and return a the number of objects found. Return result as a pointer to a *`QueryResult`* object.
status	Return status (after constructor) for error checking.
name, longName, shortName	Return the name (longName) or shortName of this catalog.
feedback	Arrange to have feedback information written to a given FILE while waiting for getImage to complete (used to display a progress bar in a user interface).

3.1.6 Error Handling

Error handling consists mainly of two default library routines defined in the *tclutil* package: one for normal errors and one for system related errors that should include the system error text:

```
error(const char* msg1, const char* msg2, int code = 0);
sys_error(const char* msg1, const char* msg2);
```

These routines are defaults that can be overridden by the application if they are defined there. The default versions for Tcl applications generate Tcl errors while the default versions for plain C/C++ applications print the messages on the stderr.

In any case, the error reporting routines keep a copy of the most recent error message, which can be retrieved using the getError() routine/method.

C++ Error Reporting Routines

Routine	Description
error	Report the error.
fmt_error	Same as `error`, but with a *printf* like interface.
sys_error	Report the error with the system error code, like perror.
fmt_sys_error	same as `sys_error`, but with a *printf* like interface.
log_message	Used to log a message that is not an error (not used here).
last_error	Return the text of the previous error message.
last_error_code	Return the error code for the previous error.
clear_error	Reset the last_error buf to empty.
set_error_handler	Set a routine to be called when errors occur and return a pointer to the previous handler.
print_error	Simple default error handler: print the error message on stdout.

See the man page for *error(3)* in the *tclutil* package for details on the above routines.

3.2 Tcl Interface

The Tcl interface is probably the most flexible and easy to use, since you don't have to deal with classes or structures. The interface consists of two new Tcl commands: *astrocat* and *astroimage*. These commands behave much like [incr Tcl] classes in that you create a catalog object and then use it to access catalog operations.

3.2.1 Working with Catalogs in Tcl

The following code creates an `astrocat` object named `cat,` opens the GSC catalog and prints out the headings followed by the rows retrieved from the catalog:

```
astrocat cat
cat open “gsc@eso”
puts "catalog headings: [cat headings]"
foreach i [cat query -pos {3:19:48 +41:30:39} -radius 10 -nrows 10] {
    puts $i
}
cat close
```

The `astrocat` command creates a new Tcl command with the same name as its argument. The new command can be used to perform operations on catalogs. A new catalog is opened with the `open` subcommand, which takes the name of the catalog as an argument. The `headings` subcommand returns a Tcl list of the catalog's column names, which can be used to display query results in a table. The `query` subcommand is used to perform the catalog queries. It takes a number of option/value pairs as arguments and returns a Tcl list of astronomical objects as the result. The format of the query subcommand is:

```
<catObject> query -option value -option value ...
```

The available options are listed below:

Tcl AstroCat Query Subcommand Options

Option	Value
-pos	Center position as a list of {ra dec} for World Coordinates or {x, y} for pixel coordinates. May also be 2 positions, indicating an area, in the format {ra1 dec1 ra2 dec2} or {x1 y1 x2 y2}.
-equinox	Equinox for the position (default 2000.0).
-id	Object Id, returned from a previous query.
-nameserver	Name server to use for resolving object names (for example simbad@eso or ned@eso - default is simbad@eso).
-name	Astronomical object name to be resolved by a name server.
-radius	Max radius from position/name (or list of min and max radius) in arcmin.
-nrows	Max number of rows to return.
-mag	Max magnitude of object (or list with min and max magnitude).
-columns	List of columns to return, default: all columns returned by server.
-sort	List of columns to sort by.
-sortorder	Specify the sort order as “increasing” or “decreasing”
-searchcols, -minvalues, -maxvalues	The argument for `-searchcols` is a list of columns to search by. The `-minvalues` and `-maxvalues` options supply the corresponding value ranges and must have the same lengths.

Some of the query options will take a Tcl list of two items as an argument, in which case a range search will be made. For example, the command:

```
cat query -pos {3:19:48 +41:30:39} -radius {10.5 20.2} -mag {2.3 4.5}
```

returns a Tcl list of objects between radius 10.5 and 20.2 from the given position and with a magnitude between 2.3 and 4.5.

The table below summarizes the `astrocat` subcommands. See the man pages for a detailed description.

Astrocat Tcl Subcommands

Subcommand	Description
check	Check that the given filename is a valid local catalog (tab table format).
checkrow	Check that the given row (tcl list of column values) is valid for the catalog (use before inserting a new row).
close	Close the current catalog.
copyright	Return the copyright field from teh catalog config entry.

Astrocat Tcl Subcommands

Subcommand	Description
id_col, ra_col, dec_col, x_col, y_col	These subcommands return the column index for the id, ra, dec, x or y column, from the catalog config entry.
entry	This subcommand is used to manipulate (get, set, update) catalog config entries.
feedback	Specifies a Tcl file descriptor to use to write feedback info during HTTP transfer of catalog data.
getcol	Given a row of output from a query, return the value for the named column in Tcl
getidpos	Given a row of output from a query, return a list *{id ra dec}*, or *{id x y}* if we're not using World Coordinates, for the given row.
getimage	Given a preview URL from a query output line, get the preview image, put it in a temp file and return the name of the file.
getpreview	Given a URL from a query output line, get the preview data and put it in a temp file and return a list of {filename type}, where filename is the name of the file containing the preview data and type is either "`image`" for a FITS image or "`table`" for a tab table with data to be plotted in a graph.
hascol	Return true if the catalog has the given column.
headings	Return a Tcl list of catalog column headings.
info	Returns a list of catalogs of a given type and/or under a given catalog directory/
is_tcs	Returns true if the catalog is a TCS catalog (flag set in config entry).
iswcs ispix	*iswcs* returns true if the catalog supports World Coordinates, *ispix* returns true if image pixel coordinates are supported. If both of these return false, no coordinates are supported. By default, catalogs are assumed to support World Coordinates.
load	Loads the named catalog config file, making the catalogs in it available to the application.
longname	Return the long name of the current or given catalog.
more	Return true if there were more than `nrows` rows available.
open	Open the named catalog.
plot	Plot the contents of the given tab table file in the given BLT graph widget.
query	Query the catalog and return a Tcl list of result rows.
querypos	Return the world or image coordinate position arguments from the most recent query, posibly expanded by a name server such as SIMBAD.
reload	Reloads the default catalog config file and updates the internal catalog entries from it.
remove	Remove specified objects from a local catalog file.

Astrocat Tcl Subcommands

Subcommand	Description
root	Returns the name of the root catalog directory.
save	Save query results to a file as a local catalog or insert into existing local catalog.
searchcols	Set or get the list of search columns in the catalog config entry.
servtype	Return the serv_type field from the catalog config file, or “local” for local catalogs.
shortname	Return the short name of the catalog.
showcols	Set or get the list of columns to display in the catalog config entry.
sortcols	Set or get the list of sort columns in the catalog config entry.
sortorder	Set or get the sort order in the catalog config entry.
symbol	Return the name of symbol used to plot the catalog objects.
url	Return the URL for the current or given catalog.

3.2.2 Image Servers in Tcl

The `astroimage` Tcl command is used to access image servers. The format of the command is similar to the `astrocat` command. For example, the following code retrieves an image file from the DSS server:

```
astroimage cat
cat open dss@eso
set filename [cat getimage -pos {3:19:48 +41:30:39} -width 10 -height 10]
```

The `astrocat` command creates a new Tcl command with the same name as its argument. This command can be used to access image server commands. The `open` subcommand is used to connect to an image server. To retrieve and image, you need to specify the position as a name or in world coordinates and a width and height for the image. The table below summarizes the available options for the getimage subcommand:

Getimage Subcommand Options

Option	Value
-pos	World coordinates of center position, format: H:M:S[+-]D:M:S.
-equinox	Equinox for the position (default 2000.0).
-nameserver	Name server to use for resolving object names (for example simbad@eso or ned@eso - default is simbad@eso).
-name	Astronomical object name to be resolved by a name server.
-width	Width of image in arc minutes.
-height	Height of object in arc minutes.

The return value from the `getimage` subcommand is the name of a temporary file holding the FITS image for the requested area. The file should be copied if necessary, since it will be overwritten by subsequent calls to `getimage`. The table below summarizes the `astroimage` subcommands:

Astroimage Tcl Subcommands

Subcommand	Description
open	Open a connection to an image server.
close	Close the current connection.
getimage	Request an image at a given position with a given width and height.
centerpos	Return the world coordinate position of the previous image request.
info	Get list of catalogs or image servers of given type.
longname	Return the long name of the image server.
shortname	Return the short name of the image server.
feedback	Specifies a Tcl file descriptor to use to write feedback info during HTTP transfer of image data.

3.2.3 Error Handling in Tcl

Errors in the Tcl interface commands are reported in the usual way, by generating a Tcl error. You can use the Tcl `catch` command to intercept errors and take any necessary action.

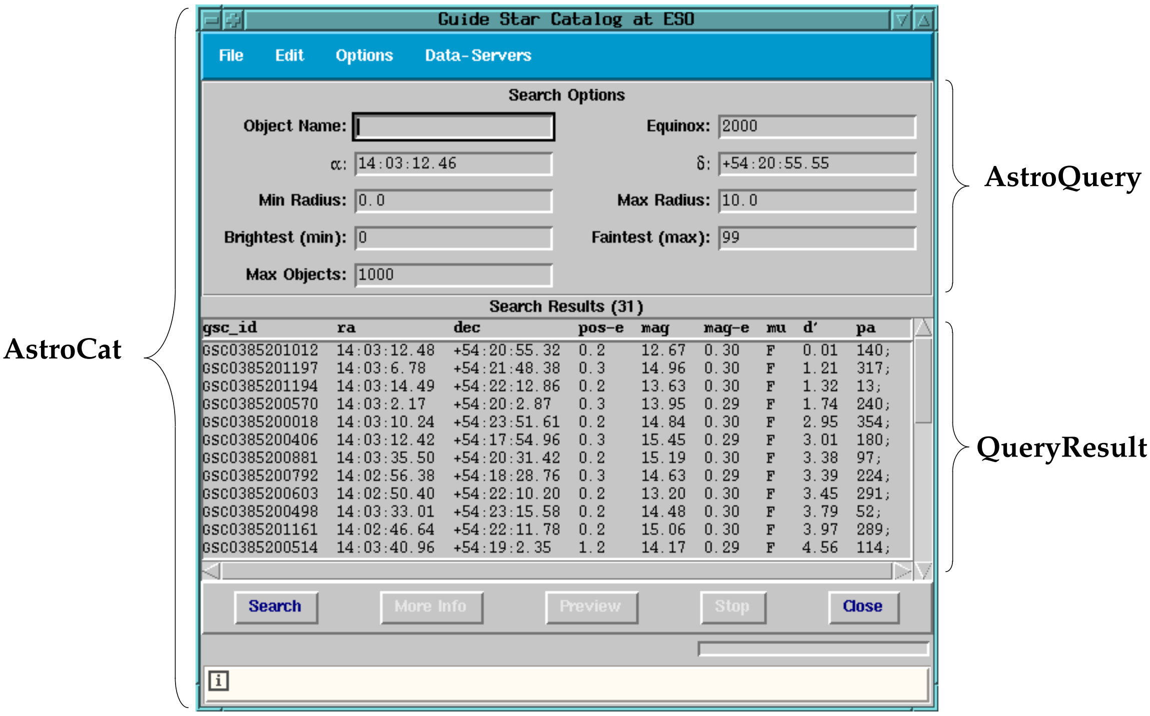
3.2.4 [incr Tk] User Interface Classes for Accessing Catalogs

The *cat* package includes a library of [incr Tk] user interface widget classes. These include widgets for making querries, displaying query results, browsing catalog hierarchies, setting options, and editing config files.

The following gives an overview of the widgets available. See the Reference section for details. Note that these classes do not deal with images or plotting objects in images. For that, see the derived widget classes in the *skycat* package.

3.2.4.1 The AstroCat, AstroQuery and QueryResult Widgets

These widgets are used for querying catalogs and viewing query results.



The *AstroCat* widget implements the top level window shown above. It combines an *AstroQuery* widget, containing the search options with a *QueryResult* widget for displaying the query results. The menubar at top and the short help window at the bottom are inherited from AstroCat's base class, *TopLevelWidget* (see the tclutil package documentation for a description of this class). The button frame at the bottom is just a Tk frame containing some buttons that access AstroCat methods.

The *AstroQuery* widget, at top, creates a frame with labels and entries for searching the catalog, based on the catalog configuration entry. The **Object Name** entry as well as the α and δ entries are only displayed if the catalog supports world coordinates, otherwise, if the catalog uses image pixel coordinates, **X** and **Y** entries are displayed. The other entries depend on the list of search columns found in the catalog configuration entry. When the user types `return` or hits the *Search* button, a caller specified command is evaluated. Below is an example of how the AstroCat class creates an instance of the AstroQuery widget:

```
method add_search_options {} {
      itk_component add searchopts {
            set searchopts_ [AstroQuery $w_.searchopts \
                    -relief groove \
                    -borderwidth 2 \
                    -astrocat [code $w_.cat] \
                    -command [code $this query_done]]
      }
      pack $itk_component(searchopts) \
            -side top -fill x
      }
}
```

The *-astrocat* option specifies the name of the C++ *astrocat* object used to access the catalog. The caller specified method, *query_done*, is called whenever a query has completed, with the following arguments:

errorMessage If this is not empty, it contains an error message and the following arguments should be ignored. If there were no errors, this arg is an empty string.

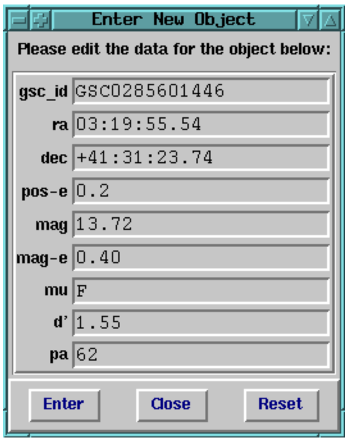
headings These are the column headings of the query result, in the form of a Tcl list.

info This is a Tcl list of rows, the result of the query, where each row is a list of column values.

more This flag is set to 1 if there were more rows available that were not returned due to the maximum row limit set.

The widget used to display the results of a catalog query is called *QueryResult*. This widget is basically a *TableList* widget (described in the *tclutil* documentation) with a few methods added for working with catalogs. The main job of this widget is to format the query results and add bindings so that the user can click on the list to do something with the data. This class also implements some methods for adding, removing, and editing rows in a local catalog using the EnterObject widget (see below).

3.2.4.2 The EnterObject Widget

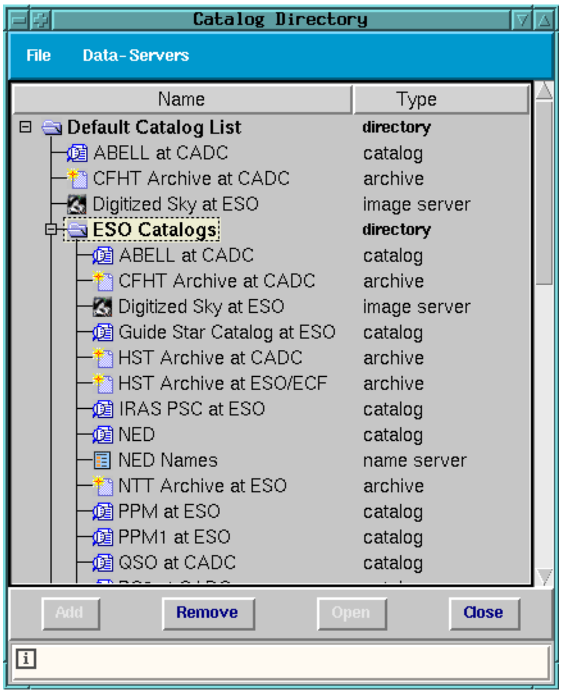


This widget is used for entering or editing catalog data. It is basically a simple *EntryForm* widget (described in the *tclutil* documentation), with some added functionality. You specify the list of column names (headings) and optionally the values, and the widget displays entries for each value.

Example:

```
EnterObject $w_.ef \
 -title {Please enter the data for a new object below:} \
 -labels $headings \
 -command [code $this enter_object]
```

3.2.4.3 The CatalogInfo Widget



This widget is used for browsing catalog directory hierarchies. It displays a hierarchical list of catalogs and catalog directories.

When first started, the window displays the default list, which is either taken from a local config file or from some HTTP server. The default configuration file is taken from *http://archive.eso.org/skycat/*, however this can be overridden with the `CATLIB_CONFIG` environment variable.

When the user clicks on a directory entry, the contents of the catalog configuration file at the given site are fetched (usually via HTTP GET) and the new catalog information is inserted in the list at the appropriate place in the list.

You can navigate the list by clicking on the + and - directory buttons. Double-clicking on a catalog entry also opens a window for the catalog. You can add and remove catalog entries using the interface and save the result to a local catalog config file that can then be used as the new default catalog list.

Example:

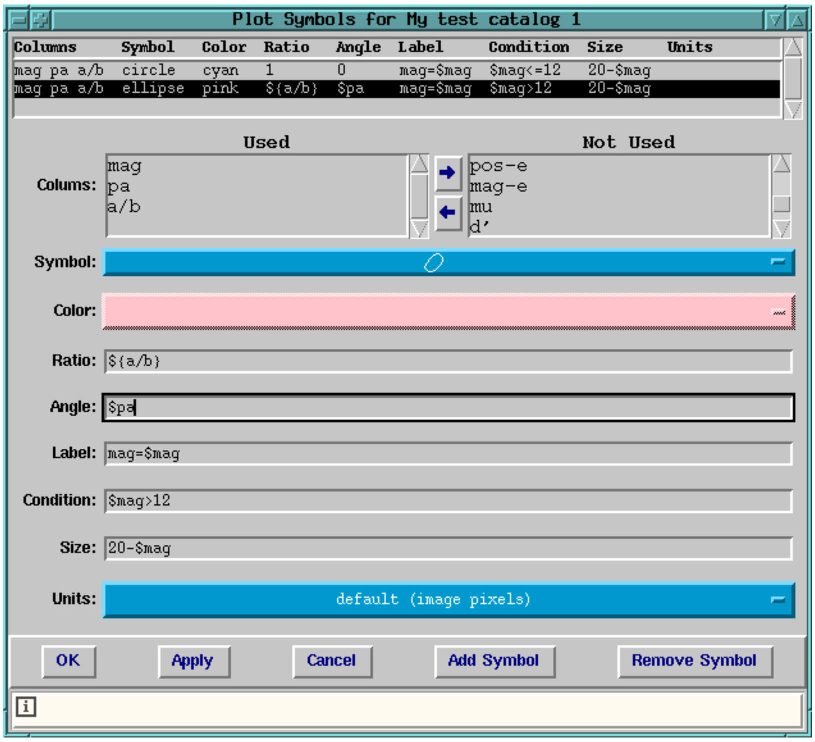
```
CatalogInfo .catinf \
    -id $id \
    -classname $classname\
    -debug $debug \
    -command [code AstroCat::update_catalog_menus]
```

The example above creates a CatalogInfo window. This code is implemented in an *AstroCat* method, *catalog_directory*.

```
catalog_directory $id $classname $debug
```

The arguments are all optional: *$id* is a unique string used when creating new catalog windows (*skycat* uses this to pass the image name, for example). *$classname* is the name of the class used to create a new catalog window (defaults to *AstroCat*). This can be overridden to use a derived class. *$debug* is a flag that is also passed to any new catalog windows created. The *-command* argument specifies a command to be evaluated whenever the list of catalogs changes. In the above example, this is used to update the catalog menus.

3.2.4.4 The SymbolConfig Widget



This widget is used for setting plot symbol options. The information goes into a catalog configuration file and can be used by applications to plot catalog objects over an image.

Each row in the table at the top of the window describes one symbol to be plotted for each object in the catalog matching the given *condition*. All of the type-in entries may contain Tcl style expressions using selected column names as variables. These variables are set for each row in the results of a catalog query before plotting any symbols.

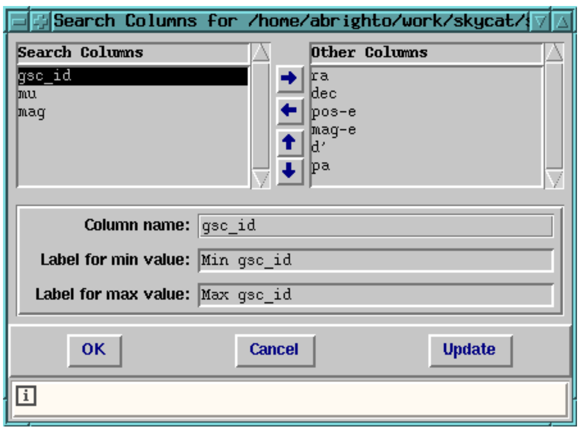
You do need to indicate which column names you plan to use as variables in the **Used** list at the top of the frame.

Example:

```
set_plot_symbols
```

The *AstroCat::set_plot_symbols* method creates this window. The *AstroCat::plot* method is called whenever the information is changed to update the plot window. Since the *AstroCat* class does not actually do any plotting, this method must be defined in a derived class to do plotting. See the *skycat* package for an example of this.

3.2.4.5 The SearchConfig Widget



This widget is used for setting or editing the list of search columns for a catalog. The listbox on the right side contains all of the column headings for the catalog. You can use the arrow buttons to move column headings to the left listbox or rearrange them there. For each name in the list, you can specify the labels to appear in the *AstroCat* (or derived class) widget. The default labels are "**Min** ..." and "**Max** ..." (replace ... with the column name).

When a query is made, the values the user typed in at these labels are made available through the **%cond** variable in the catalog query URL. This variable expands to a string of the form:

```
colname1=minValue,maxValue&.colname2=minValue2,maxValue2&...
```

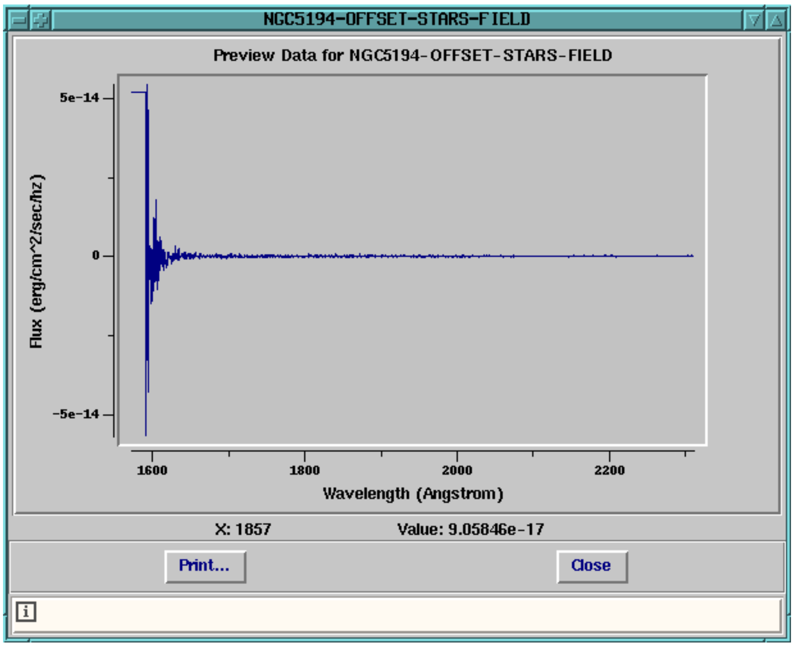
So, for example, if the catalog configuration file contained the following line:

```
url: http://archive.eso.org/skycat/servers/foo?%ra%dec%cond
```

The resulting URL might look like this (only the entries with non-empty values are used):

```
http://archive.eso.org/skycat/servers/foo?03:22:41-37:12:36&mag=0,16
```

3.2.4.6 The PreviewPlot Widget



The *PreviewPlot* widget is used to display a graph of data returned from the special *Preview* catalog columns. Some catalogs (such as the HST catalogs) return a column containing a URL that points to data that can be retrieved and plotted in a graph. By convention, the column name is `Preview` (case insensitive).

The format of the data is a tab table (the same as the local catalog format) with 2 columns, one for each axis. The column headings are displayed as the X and Y axis labels.

This widget is created automatically by the *AstroCat* class when the **Preview** button is pushed (in method *get_preview*).

3.3 C Interface Library

The catalog library also provides a simple C interface, for use in C based applications. Note, however, that C applications that use C++ libraries must have a C++ main and be linked with the C++ compiler. The simplest thing to do is to rename your C main routine to c_main and call it from a one line C++ main.

3.3.1 Information Catalogs in C

To use the C interface to access an information catalog, you should first include the header file:

```
#include "astroCatalog.h"
```

Then, to open a catalog, you specify the catalog name:

```
AcHandel cat = acOpen(catName);
if (!cat) {
      error(...);
}
```

The return value from `acOpen` is a handle that can be used in further calls to refer to the catalog. The catalog name should be one of the valid names listed in the catalog config file[1]. When you are done with the catalog, you can close it and free any resources used by with the `acClose` routine:

```
acClose(cat);
```

To get a description of the columns in the catalog, you can do the following:

```
int numCols;
char** colNames;
int status = acGetDescription(cat, &numCols, &colNames);
```

1. The C++ interface has methods to get a list of catalogs and catalog attributes. This is currently missing in the C interface.

The above call passes the catalog handle as the first argument, as always. The next arguments are set by `acGetDescription` to the number of columns in the catalog and an array of column names. The return value here, as for most of the routines defined in the C interface, is 0 for success and 1 for error. If an error occurs, you can get the text of the error message with a call to acGetError:

```
if (status != 0)
      printf("%s\n", acGetError());
```

A number of routines are defined for the various different types of catalog queries. If you already have an object Id from a previous query, this call will get the attributes of the object given by the Id:

```
AcResult result = NULL;
status = acGetObject(cat, id, numCols, colNames, &result);
```

The input arguments specify the catalog handle (*cat*), object *Id* (string value) and an array of column names (of length *numCols*) for the columns that should be returned. The return argument (*result*) is a handle, similar to the catalog handle, that can be used to access the results[1] of the query. Routines are defined that operate on the query result to extract row and column values in various formats and give other information about the result, such as the result column names, number of rows and columns, etc (see below for more).

To get a list of all of the stars in a given area, marked by two world coordinate points, the following routine is defined:

```
status = acGetArea(cat, numCols, colNames, ra0, dec0, ra1, dec1, mag0,
                   mag1, maxRows, filename, &numFound, &result);
```

As in the previous routine, the first three arguments specify the catalog handle, the number of columns and an array of columns names for the columns that should be returned by the query. The next four arguments specify the world coordinates of the two corner points of the area to be searched. The two magnitude arguments, mag0 and mag1, are optional and can be set to 0.0 if not needed. Otherwise, they specify the min and maximum magnitude for the search. The maxRows argument limits the number of rows returned by the query. The `acMore`() routine can be called to determine if there would be more objects matching the query. If `filename` is not NULL, the results of the query are written to the given file, otherwise, the results are returned in the last argument.

Other query routines are: `acCircularSearch` and `acSearchClosestStar`, with a similar interface:

```
status = acCircularSearch(cat, numCols, colNames, ra, dec,
      radius0, radius1, mag0, mag1, maxRows, filename, &numRows, &result);
```

In this case, one world coordinate position and two radius values are specified for a circular search. The objects returned will be a distance between radius0 and radius1 from the center position. If mag0 and mag1 are not null (0.0), they are taken as the min and max magnitude. The rest of the arguments are the same as for `acGetArea`().

To find the star closest to a given point, use `acSearchClosestStar`:

```
status = acSearchClosestStar(cat, colNames, ra, dec, mag0, mag1,
      &numRows, &result);
```

Here you specify one world coordinate point and again the min and max magnitude (or 0.0, to ignore) and the return value is the handle for accessing the query result.

1. Note that memory is allocated for the query results and it is the caller's responsibility to delete the object with the acrDelete() routine when no longer needed.

The following table summarizes the C routines for querying information catalogs:

AstroCatalog C Routines

Routine	Description
acOpen	Open a catalog and return a handle for it.
acClose	Close the catalog and free any resources used by it.
acGetDescription	Return a description of the catalog columns.
acGetObject	Return the column values for a given object Id.
acCircularSearch	Return a table of objects in a given circular area.
acSearchClosestStar	Return the column values for the closest object to a point.
acGetArea	Return a table of objects found in a given rectangular area.
acGetError	Return the text of the most recent error message.
acGetErrorCode	Return the error code from the most recent error message (a value from <errno.h>).
acMore	Return true if there were more than *nrows* rows available.
acNumCols	Return the number of columns in the catalog.
acColName	Return the column name given the index.
acColIndex	Return the index for a given column name.
acIsNumeric	Return true if the given column is numeric.

3.3.1.1 Accessing Query Results

The last argument to the catalog query routines is a *handle* of type *AcResult*. Routines are defined to access the query result by specifying a row and column index or column name and a pointer to a variable to hold the column value. For example, to print out the query results:

```
for (i = 0; i < numRows, i++) {
      for (j = 0; j < numCols; j++) {
            char* s;
            if (acrGetString(result, i, j, &s) == 0)
                  printf(“row %d, col %d: %s\n”, i, j, s);
            else
                  error(...);
      }
}
```

In this case, we treated all of the items as strings. Routines are also defined to access row,column values as int, short, char, float, double and world coordinates. To get the value for the column “mag” in a given row:

```
double d;
if (acrGetNDouble(result, row, “mag”, &d) == 0) {/* OK ...*/}
```

To get a world coordinate position, you can use the following:

```
WC pos;
acrGetWC(result, row, &pos);
```

Where WC is a structure (described in the previous section) that represents world coordinates.

There are also routines to get the names of the result columns as well as the number of rows and columns in the result. The following table summarizes the routines for accessing query results:

C Routines to Access Query Results

Routine	Description
acrNumRows	Return the number of result rows.
acrNumCols	Return the number of result columns.
acrColNames	Return a pointer to an array of result column names.
acrGetString acrGetInt acrGetDouble acrGetFloat acrGetShort acrGetChar	Get result values by row and column index. Note: there are various versions for different data types and parameters. All return 0 for success and set the last argument value or return 1 for error.
acrGetNString acrGetNInt acrGetNDouble acrGetNFloat acrGetNShort acrGetNChar	Get result values by row and column name. Note: there are various versions for different data types and parameters. All return 0 for success and set the last argument value or return 1 for error.
acrGetWC	If the result contains a world coordinate position (fields ra and dec), get it and return success (0), otherwise error (1).
acrColIndex	Return the result column index for the given result column name.
acrDelete	Delete the result object (free the memory).

3.3.2 Image Servers in C

The AstroImage server interface is similar to the catalog interface. To use it, you should include the `astroImage.h` header file:

```
#include "astroImage.h"
AI_Handle im = aiOpen(imageServerName);
if (!im) {
      error(...);
}
...
filename = aiGetImage(im, ra_center, dec_center, width, height);
if (filename == NULL) {
      error(...);
}
```

The `aiOpen` routine returns the handle for the image server. The `aiGetImage` routine takes as parameters the handle, the world coordinates (in degrees, as doubles), the width and height in arc minutes of the image to be retrieved. The return value is the name of a temporary file containing the FITS image retrieved from the server. The file should be copied if necessary, since it will be rewritten in the next call to `aiGetImage`.

4 REFERENCE

Following is a list of man pages for the various classes and utilities provided in the *cat* package.

4.0.1 AstroCat(n)

NAME

AstroCat - user interface class for viewing catalog info

NAMESPACE

cat

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

AstroCat <path> ?options?

DESCRIPTION

The AstroCat widget class defines a top level widget for searching and displaying astronomical catalog data. It contains a menubar with items for loading, editing, and saving catalog data and a table displaying rows and columns of catalog data. This class does not know anything about images or plotting objects in images, however these features may be added in a derived class (see the SkySearch(n) in the skycat package for an example).

You can also run this class as a standalone application with the command "astrocat". The application options are the same as the class options.

OPTIONS

-anchor

Set the anchor for labels.

-catalog

Name of catalog.

-catalogtype

Type of catalog (catalog or archive).

-debug

Flag: if true, run queries in foreground for better debugging.

-id

Optional unique id, used in searching for already existing catalog widgets.

-labelfont

Font to use for labels.

-labelwidth

Set the width for displaying labels.

-tcs

If -tcs is 1, use fixed format TCS catalog listings.

-tmpfile

Temp image file to use for preview.

-valuefont

```
        Font to use for values.

-wcsfont
        Font to use for ra,dec labels.
```

METHODS

```
add_dialog_buttons {}
        Add the dialog button frame.

add_menubar {}
        Add the menu bar.

add_progress_bar {}
        Add a progress bar to display the progress of data transfers.

add_result_table {}
        Add the table for displaying the query results.

add_search_options {}
        Add the search options panel.

add_selected {}
        Add the currently selected rows to a local catalog file.

add_to {}
        Add the rows in the current listing to a local catalog file.

clear {}
        Clear the table listing.

close {}
        Close this window.

display_image_file {filename}
        Display the given image file (To be defined in a subclass).

edit_selected_object {}
        Pop up a window so that the user can edit the selected object(s).

enter_new_object {}
        Pop up a dialog to enter the data for a new object for a local
        catalog.

feedback {}
        This method is called by the fileevent handler during the data
        transfer from the HTTP C++ class to give the user feedback about
        how much there is left to copy, etc...  Read a line from the
        feedback pipe, which should contain text to be displayed in the
        progress widget.

        When we are in busy (disabled, waiting) mode, we also interpret
        some special messages, such as: "total length: n bytes" to update
        the progress bar.

get_col {name row}
        Return the value for the given column name in the given row.

get_name_servers {}
        Add the name server catalogs to the given menu.

get_preview {}
        Get the preview image given by $preview_url_ into a temp file.
```

```
init {}
        Called after options have been evaluated.

interrupt {}
        Interrupt the current search .

layout_dialog {}
        Do the dialog window layout.

make_short_help {}
        Add a short help window and set the help texts.

more {}
        Use the "more" URL to display more info about an object using
        netscape or mosaic.

open_catalog {}
        Open the catalog for this window.

plot {}
        Plot the stars/objects found in the previous search in a (image)
        window.  The symbols to use are taken from the config file.  (To
        be defined in a subclass).

preview {}
        Assuming there is a field called "preview" that is the URL of an
        image, get the image or table data for the selected line over HTTP
        and display it

        The preview data may be an image or other data, such as a tab
        table to display as a graph (like the "cuts" (RtdImageSpectrum)
        graph).

preview_done {status result}
        This method is called when we have received the preview data The
        "status" argument is the status of the background http get
        operation (0 if ok).  The result is a list of {filename
        Content-type}, where filename contains the data and Content-type
        indicates the type of the preview data. Decompression is already
        taken care of, so we only expect to get a FITS image here or a tab
        table to plot.

print {}
        Pop up a dialog to print the table listing.

query_done {errmsg headings info more}
        This method is called when the background query is done.

        Args:

          errmsg - If this is not empty, it contains an error message and
          the following args should be ignored. If there were no errors,
          this arg is an empty string.

          headings  - are the column headings

          info - is a list of rows (result of query)

          more - is a flag set to 1 if there were more rows available that
          were not returned due to the maxrows limit set.

quit {}
```

```
        Quit the application (for standalone version).

remove_selected {}
        Remove the currently selected rows from a local catalog file.

reset_table {}
        Reset table dialogs if needed.

save_as {}
        Save the current data to a local catalog.

search {args}
        Start the catalog search based on the current search options and
        display the results in the table.

select_columns {}
        Pop up a dialog to select table columns to display.

select_result_row {}
        This method is called whenever a result row is selected note the
        selected values and enable/disable some buttons.

send_to_netscape {url}
        Send a URL to be displayed by netscape.

set_instance {}
        Keep an array of instances(name,id) to help locate the window for
        a catalog.

set_logfile {}
        Create the ~/.skycat dir if it does not already exists and keep a
        log file there.

set_menu_states {}
        Enable or disable some menus.

set_namesvr {name}
        Set the name server to use to resolve object names.

set_plot_symbols {}
        Pop up a dialog to set the plot symbols to use for this catalog.

set_pos_radius {list}
        Set the values for the position and radius entries from the given
        list, which should be in the format {ra dec equinox radius} if we
        are using wcs, otherwise {x y radius}.

set_search_cols {}
        Pop up a dialog to set the search columns to use for this
        catalog.

set_show_cols {cols}
        Called when the user has selected columns to show.

set_sort_cols {sort_cols sort_order}
        Called when the user has selected columns to sort the results by.
        The first arg is the sort columns, the second arg is the order
        (increasing, decreasing).

set_state {state}
        Set/reset widget states while busy .

set_tcs_columns {}
```

```
        Toggle the TCS option (use the tcscat or the astrocat Tcl
        command).

sort_dialog {}
        Pop up a dialog to sort the list.

update_search_options {}
        Update the search option entries after they have been edited.
```

PROCS

```
add_catalog_menu {w {id ""} {classname AstroCat} {debug 0}}
        Add a menubutton with catalog items to the given TopLevelWidget.

        w is the caller's toplevel window (an instance of
        TopLevelWidget).

        id is an optional unique id to be associated with a new catalog
        widget.

        classname is the name of the AstroCat subclass to use to create
        new catalog widgets (defaults to "AstroCat").

        debug is a flag is passed from the command line. If true, querries
        are run in the foreground, to make debugging easier.

        note: this is defined as a proc rather than a method so that it
        can be accessed from outside the class to add the Data-Servers
        menu.

catalog_directory {{id ""} {classname AstroCat} {debug 0}}
        Pop up a window to browse the catalog directories.

        id is an optional unique id to be associated with a new catalog
        widget.

        classname is the name of the AstroCat subclass to use to create
        new catalog widgets (defaults to "AstroCat").

        debug is a flag is passed from the command line. If true, querries
        are run in the foreground, to make debugging easier.
        .

fill_catalog_menu {w id classname m debug serv_type {cmd longname}}
        Fill up a menu of known data servers to choose from.

        w is the caller's top level widget (containing menubar)

        id is an optional unique id to be associated with a new catalog
        widget.

        classname is the name of the AstroCat subclass to use to create
        new catalog widgets (defaults to "AstroCat").

        $debug is a flag set from the command line arg.

        The serv_type argument should be one of: "catalog", "archive".

        An optional "cmd" argument may be set to longname or shortname
        depending on the names you want in the menu (for local catalogs,
        specify shortname, to avoid getting the complete path name in the
        menu.
```

```
instances {}
        Return a list of the instances of this class.

is_tcs_catalog {filename}
        Return true if the given file contains a TCS catalog.

list_windows {}
        Return a list of top level windows that this class might create.

local_catalog {{id ""} {classname AstroCat} {debug 0}}
        Ask the user for the name of a local catalog file and then open a
        window for the catalog.

        id is an optional unique id to be associated with a new catalog
        widget.

        classname is the name of the AstroCat subclass to use to create
        new catalog widgets (defaults to "AstroCat").

        $debug is a flag set from the command line arg.

new_catalog {name {id ""} {classname AstroCat} {debug 0} {tcs_flag 0}
            {type "catalog"}}
        This member proc is used to open a window for the named catalog,
        or reuse the existing one for the catalog, if it is already open.

        name is the long name of catalog from config file

        id is an optional unique id to be associated with a new catalog
        widget.

        classname is the name of the AstroCat subclass to use to create
        new catalog widgets (defaults to "AstroCat").

        debug is a flag: if true, run queries in foreground

        tcs_flag may be set to true if the catalog is TCS format

        type is the catalog serv_type.

reload_config_file {w}
        Reload the default catalog config file using the given astrocat
        object.

select_catalog {name serv_type id classname tcs_flag debug}
        This proc is called when the named catalog selected from the
        menu.

        serv_type is the type of the catalog: one of: catalog, archive,
        local, imagesvr, etc...

        id is an optional unique id to be associated with a new catalog
        widget.

        classname is the name of the AstroCat subclass to use to create
        new catalog widgets (defaults to "AstroCat").

        tcs_flag can be set to true, if you know it is a TCS catalog.

        $debug is a flag set from the command line arg.

update_catalog_menus {}
        Update all of the catalog menus in all instances to show the
```

current catalog info. .

PROTECTED VARIABLES

col_
    Array(uppercase col name) of col index from catalog headings.

file_menu_
    Name of File menu widget.

headings_
    List of catalog column headings (from results of most recent query).

info_
    Result from most recent query (list of rows).

instance_idx_
    Index of this object in the instances_ array.

logfile_name_
    Log file handle (used to log URLs).

more_url_
    More-info URLs for selected object.

object_name_
    Currently selected object (Id field in row).

results_
    QueryResult widget used to display search results.

rfd_
    Pipe to read feedback during image transfer.

searchopts_
    AstroQuery widget used to manage search options.

COMMON CLASS VARIABLES

astrocat_
    C++ astrocat object used by static member procs.

catalog_menu_info_
    Array(TopLevelWidget instance) of command used to update the Data-Servers menu. This is used to make it posible to update all instances of this menu in various top level windows.

count_
    Count used for filename generation.

current_instance_
    Current instance name.

instances_
    Array mapping catalog name to widget/class name.

n_instances_
    Instance count.

SEE ALSO

TopLevelWidget(n)

```
- - - - - -
Last change: 02 Feb 98
```

4.0.2 astrocat(n)

NAME

```
astrocat - Tcl interface to the catalog library
```

SYNOPSIS

```
astrocat $instName
$instName subcommand ...
```

DESCRIPTION

```
The "astrocat" Tcl command is the Tcl interface to the catalog
library, in particular the AstroCatalog(3) class. This command can be
used to query catalogs and get configuration about the available
catalogs.  See the man page for AstroCatalog(3) for details that are
not described here.

The astrocat Tcl command creates a new Tcl command with the same name
as its argument ($instName) that provides the subcommands described
below.
```

SUBCOMMANDS

```
$instName delete
        Use the "$instName delete" subcommand to free the resources
        for this object and remove the Tcl command.

$instName check $filename
        Check that the given filename is a valid local catalog
        (tab table format).

$instName checkrow $row
        Check that the given row (Tcl list of columns) is valid for a
        local catalog. The row should contain valid values for ra and
        dec, if we are using world coords, or x and y, if we are using
        image coords. The default columns are "id ra dec ...", but can
        be overridden in the catalog config file or header.

        This command generates an error if the position (ra, dec) or
        (x, y) is not found or is not in range.

$instName close
        Close the current catalog, if one was open.

$instName copyright
        Return the copyright string for this catalog from the catalog
        config file.

$instName dec_col
        Return the value of the dec_col keyword for this catalog from
        the catalog config file.

$instName entry get ?$name? ?$directory?
$instName entry set $$info ?$name? ?$directory?
$instName entry update $info ?$name? ?$directory?
$instName entry add $info ?$directory?
$instName entry remove $name
        "entry get" returns the catalog config entry for the currently
        open catalog or for the given catalog if one is specified.
        The format of the return value is a tcl list of {keyword
        value} pairs:
```

```
        {{key value} {key value} ...}

        for example:

        {{serv_type catalog}
         {long_name "Guide Star Catalog at ESO"}
         {short_name gsc@eso}
         {url http://archive.eso.org/...}
         {symbol ...}
         {id_col 0}
         {ra_col 1}
         {dec_col 2}
        ...}

        If the "directory" argument is specified, the entry is searched for
        starting in the given catalog directory, otherwise it is searched for
        starting at the top level directory (the catalog entries are linked
        in a hierarchical list or directory structure).

        "entry update" allows you to update fields in the catalog's config
        entry.  Some fields, such as the catalog name and URL can not be
        changed in this way, however the column positions id_col, ra_col,
        dec_col, etc... may be updated.  The format for the update info is the
        same as that returned by "entry get". The directory argument has the
        same meaning as with the "entry get" subcommand.

        Note: The "entry update" command is needed, for example, when a
        catalog query was run in a subprocess. The results of the query may
        include config info, such as the default plot symbol and the columns
        for id, ra and dec. This subcommand can be used to get this
        information back into the parent process catalog entry.

        "entry add" adds a new catalog entry to the internal list. The format
        of the argument is the same as that for update. The entry is added to the
        given directory or the top level if none is specified.

        "entry remove" removes the entry for the named catalog from the internal
        default catalog list.

$instName feedback $fileDesc
        Specifies a Tcl file descriptor to use to write feedback info
        during HTTP transfer of catalog data.  1 Arg: file descriptor.

$instName getcol
        Given a row of output from a query, return the value for the
        named column in Tcl.

        example: set preview [$cat getcol preview $row]

$instName getidpos
        Given a row of output from a query, return a list {id ra dec}
        (or {id x y} if we're not using wcs) for the given row. The
        positions of the 3 columns default to the first 3 columns: id,
        ra and dec, but might be specified differently in the catalog
        config entry or result header.

        example: lassign [$cat getidpos $row] id ra dec

$instName getpreview ?-url $url? ?-tmpfile $filename?
        Given a URL from a query output line, get the preview data and
        put it in a temp file.  The return value in Tcl is a list of
        {filename type}, where filename is the name of the file
```

```
        containing the preview data and type is either "image" for a
        decompressed FITS image or "table" for a tab table with data
        to be plotted in a graph.

        The temp file is re-used and deleted in the destructor.

         example: set preview [$cat getpreview -url "http://...." ?-tmpfile "/tmp/
img..."?]

        -url specifies the URL to use.

        -tmpfile can be used to specify the image file name to use.


$instName hascol
        Return true if the catalog contains the given column name (in
        the config entry).

$instName headings
        Return a Tcl list of the column headings.

$instName id_col
        Return the value of the id_col keyword for this catalog from
        the catalog config file.

$instName info $serv_type ?$directory?
        This command returns a list of catalogs from the catalog
        config file.  The "serv_type" argument determines which
        catalogs are listed (one of: catalog, namesvr, imagesvr,
        directory, local...).

        If $serv_type is an empty string, all entries are returned.

        If $directory is specified, the return list will be from the
        given catalog directory entry, retrieved via HTTP if
        needed. The default, if no $directory is given, is the top
        level list read from the default config file at startup.

$instName is_tcs ?$name? ?$newValue?
        With 0 or 1 arg, returns true if this catalog (or the given
        one) is a TCS catalog.

        If $newValue is specified, it sets the tcs flag for the given catalog.

$instName ispix
        Returns true if the catalog is based on image pixel coordinates.

$instName iswcs
        Returns true if the catalog is based on world coordinates.

$instName load $filename
        This command loads the named catalog config file, making the
        catalogs in it available to the application.

$instName longname ?name?
        Returns the long_name field from the catalog config file
        for the given long or short name, or for the current catalog.

$instName more
        Return true if there were more rows to fetch on the last query
        (i.e.: not all rows that were found were returned due to
        limits).
```

```
$instName open
        Open the given astromonical catalog and refer to it in future
        queries.

$instName plot $graph $element $filename $xVector $yVector
        Plot the contents of the given tab table file in the given
        BLT graph widget.

        element    is the BLT graph element name

        filename   is the local catalog file

        xVector    is the name of the BLT x vector

        yVector    is the name of the BLT y vector

        The data for the given element in the given graph will be set
        directly from here without going through tcl.

        The return value in Tcl is the number of rows (for the graph's
        x-axis)

$instName query ?-option $value ...?
        Pass a query to the current catalog and return the result as a
        Tcl list of rows, where each row is a Tcl list of values.

        Options and Values:

         -id $id
            Specify the catalog id of the object, (as returned from a
            previous query). If this is specified, -pos, -name, -mag
            and -radius should not be specified and will be ignored.

         -pos {ra dec}
         -pos {x y}
         -pos {ra1 dec1 ra2 dec2}
         -pos {x1 y1 x2 y2}
            World {ra, dec} or image {x, y} coordinates of center
            position, or list {ra1 dec1 ra2 dec2} or {x1 y1 x2 y2} of
            2 points for an area.  World Coordinates are given as
            {H:M:S[+-]D:M:S} in the given equinox.  If the catalog
            config entry contains the keywords "x_col" and "y_col",
            the coords are interpreted as image coords, otherwise
            world coords.

         -equinox $equinox
            specify the quinox for position (default 2000).

         -width $w
         -height $h
            Dimensions of rectangle with pos at center (alternative to
            specifying 2 positions) in arcmin for world coords or
            pixel for image coords.

         -mag $mag
             Max or list {min max} magnitude of object

         -radius $r
            Max or list (min max} radius from position (in arcmin for
            world coords or pixel for image coords).

         -nameserver $ns
            Name of nameserver catalog to use (simbad@eso,
```

```
            ned@eso,...).

         -name $name
            Can be used instead of -pos. The name will be resolved
            using the value of -nameserver (default: SIMBAD)

         -columns {col1 col2 ...}
            Specify a list of columns to return.

         -searchcols {col1 col2 ...}
             Specify a list of columns to search by. The -minvalues
             and -maxvalues options supply the corresponding value
             ranges and must have the same lengths.

         -minvalues {v1 v2 ...}
         -maxvalues {v1 v2 ...}
             Specify a list of values corresponding to the columns
             specified with the -searchcols option. The values may be
             numeric or string format, but the lists must have the
             same lengths as the one specified by the -searchcols
             option.

         -sort {col1 col2 ...}
            Set the list of column names to sort by.

         -sortorder increasing
         -sortorder decreasing
            Set the sort order.

         -nrows $n
            Set the max number of rows to return.

        Each option has one value, however, for a range or area query,
        some values can be a list, such as -radius "$rad1 $rad2" to
        give a radius range or -pos "$pos1 $pos2" to give an area.

        If -columns is not specified, all columns are assumed.
        Otherwise, if -columns is specified, the column names should
        be valid for the catalog and the result will be a list of rows
        with those columns.

        Note that not all catalogs will support sorting by all fields.

$instName querypos
        Return the world or image coordinate position arguments from
        the most recent query, posibly expanded by a name server such
        as SIMBAD. The result is a list of the form {ra dec equinox}
        for world coordinates or just {x y} for image coords.

$instName ra_col
        return the value of the ra_col field from the config file.

$instName reload
        This command reloads the default catalog config file and
        updates the internal catalog entries from it. This is meant to
        be used after the config file has been edited by hand to make
        the new data available to the application.

$instName remove $filename ?$data? ?$equinox? ?$headings?
        With 1 arg, remove the results of the previous query from the
        given file.

        If $data is specified, it should be a Tcl list of rows to be
```

```
        removed, in the format returned by the query command.

        If $equinox is specified, it is the equinox of ra and dec in
        $data (so that ra,dec can be converted to J2000 for
        comparison).

        If $headings is given, it is used as a Tcl list of column
        headings.  Otherwise the catalog headings are used, if there
        is a current catalog.

        The data rows are removed from the file, which should be in
        the form of a local catalog (tab table), such as that used by
        the starbase utilities.

$instName root
        This command returns the name of the root catalog directory.

$instName save $filename ?$iflag? ?$data? ?$equinox? ?$headings?
        With 1 arg, save the results of the previous query to the
        given file.

        If $iflag is true, the data is inserted in the file if it
        already exists and the file is sorted and duplicates removed.

        If $data is specified, it should be a Tcl list of rows to be
        saved, in the format returned by the query command.

        If $equinox is specified, it is the equinox of the ra and dec
        columns in the data (the first 3 columns are assumed to be id,
        ra and dec, unless otherwise defined in the catalog config
        entry or header).

        If $headings is given, it is used as a Tcl list of column
        headings.  Otherwise the catalog headings are used, if there
        is a current catalog.

        The data is saved to a file in the form of a local catalog
        (tab table) that can be loaded again or processed by starbase
        utilities.

$instName searchcols ?newValue?
        With no args, return the "search_cols" entry for this catalog.
        With one arg, specify a new value for the search_cols entry,
        which should be a list of the form:

        {{colName minLabel maxLabel} ...}

$instName servtype ?name?
        Return the servtype for this catalog or for the given catalog.

$instName shortname ?name?
        Returns the short_name field from the catalog config file for
        the given long name, or for the current catalog.

$instName showcols ?newValue?
        With no args, return the "show_cols" entry for this catalog.
        With one arg, specify a new value for the show_cols entry,
        which should be a list of the form: {col1 col2 ...}.

$instName sortcols ?newValue?
        With no args, return the "sort_cols" entry for this catalog.
        With one arg, specify a new value for the sort_cols entry,
        which should be a list of the form: {col1 col2 ...}
```

```
$instName sortorder
        With no args, return the "sort_order" entry for this catalog.
        With one arg, specify a new value for the sort_order entry,
        which should be a list of the form:  {col1 col2 ...}

$instName symbol ?newValue?
        With no args, return the "symbol" entry for this catalog.
        With one arg, specify a new value for the symbol entry, which
        should be a list of the form:

           {{colNames symbolExpr sizeExpr} ...}

        where colNames is a list of column names used, symbolExpr is
        one of the accepted symbol expressions, such as "circle
        ?color?" and sizeExpr is an expression (or a list of {expr
        ?units?}) giving the radius of an object.

$instName url ?name?
        Return the url for this catalog or for the given catalog.

$instName x_col
        Return the value of the x_col keyword for this catalog from
        the catalog config file.

$instName y_col
        Return the value of the y_col keyword for this catalog from
        the catalog config file.
```

SEE ALSO

AstroCatalog(3), AstroImage(3)

\- - - - - -
Last change: 02 Feb 98

4.0.3 astroCatalog(3)

NAME

astroCatalog - C library for accessing astromical catalogs

SYNOPSIS

```
#include "astroCatalog.h"

typedef void* AcHandle;

typedef void* AcResult;

AcHandle acOpen(char* name);

void acClose(AcHandle);

int acMore(AcHandle);

int acNumCols(AcHandle)

char* acColName(AcHandle, int col);

char** acColNames(AcHandle);

int acColIndex(AcHandle handle, char* name);

char* acGetError();
int acGetErrorCode();

int acGetDescription(
    AcHandle cat,
    int* numCols,
    char*** colNames);

int acGetObject(
    AcHandle cat,
    char* id,
    int numCols,
    char** colNames,
    AcResult* result);

int acGetArea(
    AcHandle cat,
    int numCols,
    char** colNames,
    double ra0,
    double dec0,
    double ra1,
    double dec1,
    double mag0,
    double mag1,
    int maxRows,
    char* filename,
    int* numFound,
    AcResult* result);

int acCircularSearch(
    AcHandle cat,
    int numCols,
    char** colNames,
    double ra,
    double dec,
```

```
    double radius0,
    double radius1,
    double mag0,
    double mag1,
    int maxRows,
    char* filename,
    int* numFound,
    AcResult* result);

int acSearchClosestStar(
    AcHandle cat,
    int numCols,
    char** colNames,
    double ra,
    double dec,
    double mag0,
    double mag1,
    AcResult* result);

int acCatalogSearch(
    AcHandle cat,
    int numCols,
    const char* const* colNames,
    int numSearchCols,
    char** searchCols,
    const char* const* minVals,
    const char* const* maxVals,
    int maxRows,
    const char* filename,
    int* numFound,
    AcResult* result);

/*
 *  --- routines for accessing the query results ---
 */

int acrNumRows(AcResult);

int acrNumCols(AcResult);

char** acrColNames(AcResult);

int acrGetString(AcResult, int row, int col, char** value);
int acrGetInt(AcResult, int row, int col, int* value);
int acrGetDouble(AcResult, int row, int col, double* value);
int acrGetFloat(AcResult, int row, int col, float* value);
int acrGetShort(AcResult,int row, int col, short* value);
int acrGetChar(AcResult,int row, int col, char* value);

int acrGetNString(AcResult,int row, const char* colName, char** value);
int acrGetNInt(AcResult,int row, const char* colName, int* value);
int acrGetNDouble(AcResult,int row, const char* colName, double* value);
int acrGetNFloat(AcResult,int row, const char* colName, float* value);
int acrGetNShort(AcResult,int row, const char* colName, short* value);
int acrGetNChar(AcResult,int row, const char* colName, char* value);

int acrGetWC(AcResult, int row, WC* pos);

int acrColIndex(AcResult,const char* colName);

void acrDelete(AcResult);
```

DESCRIPTION

This man page describes a library for accessing astronomical catalogs from C applications in a generic way.

Note: Although the library is implemented in C++, it also defines a C interface, which is described here. To use this library from C, an application must have a C++ main and be linked with the C++ compiler. This is easily done by renaming the C main to c_main and defining a C++ main that calls it.

The main purpose of this library is to provide generic access to a variety of astronomical catalogs available locally or on the network. The interface hides the details of accessing the various catalogs and shows their contents in a uniform way.

CATALOG SERVERS

This class communicates with a catalog server via HTTP requests. The list of servers and how to access them is kept in a configuration file, which is also accessed via HTTP. The default config file is:

    http://archive.eso.org/skycat/skycat.cfg

The default can be overridden by defining the environment variable SKYCAT_CONFIG to another valid HTTP URL.

LOCAL CATALOGS

If the catalog name passed to acOpen is the name of a file, it is taken to be a local catalog file in the format used by the Starbase utilities. A local catalog is an ASCII file containing an optional title and header information followed by the column headings, a dashed line and the data rows. The columns are separated by tabs. This is also the format of query results returned via HTTP from remote catalog servers.

Example:

```
Table1
id      ra      dec     pos-e   mag
------  --      ---     -----   ---
A00050  3:19:28 8:26:29 0.2     13.49
A00098  3:19:32 8:34:15 0.4     13.40
A00288  3:19:21 8:31:19 0.2     13.13
A00314  3:19:44 8:30:58 0.2     13.98
```

The pointer returned from acOpen for files is a pointer to an object for accessing local catalogs. The interface is the same for remote and local catalogs, only the implementation is different.

QUERY PARAMETERS

Parameters for catalog searches are checked to make sure they are valid and in the correct range. A radius value must be between 0.0 and 300.0 arcmin. A magnitude may have any value. For ranges, such as min and max radius and min and max magnitude, the order of the arguments is not important, since they will be rearranged as needed. If both min and max values are 0.0, they are ignored for that search. If you really want to search for objects with "0.0" magnitude, specify a range with small values that are not zero, for example, min/max mag = (-0.0001, +0.0001).

FORMAT OF QUERY RESULTS

Query results are accessed by a set of routines that operate on a an "AcResult" object that is returned as the last argument in the query routine calls. The object keeps a table of result rows and columns internally and provides an interface for extracting the values in various formats and data types. The format of query results in a file was described above (see LOCAL CATALOGS).

ROUTINE NAMES

Note that the astroCatalog routine names all have the prefix "ac", while the routines that operate on the query results all have the prefix "acr".

MEMORY MANAGEMENT

Memory is allocated for the query results. It is the caller's responsibility to delete the result object with the acrDelete() routine when it is no longer needed.

ERROR HANDLING

Most of the routines described here return non-zero on error and zero if all is OK (routines that return pointers return a NULL pointer on error). The text of the error message can be retrieved with the acGetError() function and an error code is available via the acGetErrorCode function. The error codes are the same as those in <errno.h> and have the same meanings (EINVAL means invalid operand - or parameter, 0 means no code was available).

UNITS

Unless otherwise stated, the units for all radius values are in arcmin. Floating point values for RA and DEC are always in degrees. The default equinox is J2000.

ASTRO CATALOG C ROUTINES

```
acOpen
        Open the named catalog and return a handle for it. The handle
        can be used in subsequent operations on the catalog. If the
        catalog can not be opened, NULL is returned and an error
        message can be retrieved with acGetError().

        Arguments:
                name (in) - Catalog name, (long or short name) from the
                            catalog config file.
        Return value:
                Catalog handle for use in future calls. Use acClose() to
                free resources when no longer needed.

acClose
        Close the given catalog and free its resources.

        Arguments:
                handle (in) - Catalog handle returned from acOpen().
        Return value:
                None

acGetDescription
```

```
        Get the number of columns and the column names for the given
        catalog and return 0 if all is OK. The memory for the return
        arrays is managed internally and should not be modified by the
        caller.

        Arguments:
                handle   (in ) - Catalog handle returned from acOpen().
                numCols  (out) - Number of catalog columns.
                colNames (out) - Pointer to array of column names.
        Return value:
                If successful, 0, otherwise 1.


acGetObject
        Get the values for the specified columns for the object given
        by "id" in the catalog and return 0 if there are no
        errors. The id should be a valid object id obtained by a
        previous catalog query.  If colNames is not NULL, it should be
        an array of "numCols" column names to fetch, otherwise all
        columns are fetched.  "result" is set to point to an AcResult
        object for accessing the catalog values found (it may be
        empty). The memory for the result is allocated (see MEMORY
        MANAGEMENT above).

        Arguments:
                handle    (in ) - Catalog handle.
                id        (in ) - Object id.
                numCols   (in ) - Number of column names specified
                colNames  (in ) - Array[numCols] of column names to
                                  read, or NULL to read all columns.
                result (in/out) - Handle for accessing query results.
        Return value:
                Returns 0 on success, 1 for errors. It is not considered
                an error if no object is found. Use acrNumRows(result)
                to determine if an object was found.
        Comment:
                An error is returned if the catalog does not support
                query by Id. The GSC server now supports this, but
                older versions of the catalog config file do not have
                the new URL for this yet.

acGetArea
        Get the values for all objects in the rectangular region given
        by the two world coordinate points. If mag0 and mag1 are not
        0.0, they are taken to be the minimum and maximum magnitude
        values for the query. If colNames is not NULL, it should be an
        array of numCols column names to fetch, otherwise all columns
        are fetched. The number of object rows returned is limited to
        maxRows (the acMore() routine can be called to see if more
        objects would have been available).
        On return, numFound is set to the number of objects found and,
        if any were found, "result" is set to point to an AcResult
        object for accessing the catalog values found. The memory for
        the result is allocated (see MEMORY MANAGEMENT above).
        If filename is not NULL, the results will be copied to the
        given file in the format of a local catalog.

        Arguments:
                handle    (in ) - Catalog handle.
                numCols   (in ) - Number of column names specified.
                colNames  (in ) - Array[numCols] of column names to
                                  read, or NULL to read all.
                ra0       (in ) - World coordinates of first point
```

```
                dec0      (in ) - in degrees.
                ra1       (in ) - World coordinates of second point
                dec1      (in ) - in degrees.
                mag0      (in ) - Min magnitude.
                mag1      (in ) - Max magnitude.
                maxRows   (in ) - Max number of rows to return.
                filename  (in ) - If not null, write results to this
                                  file.
                numFound  (out) - Set to number of objects found.
                result (in/out) - Handle for accessing query results.
        Return value:
                If successful, 0, otherwise 1.
        Comment:
                An error is returned is the catalog does not support
                area querries.

acCircularSearch
        Get the values for all objects in the circle or ring given by
        the world coordinate point and the min and max radius values.
        If mag0 and mag1 are not 0.0, they are taken to be the minimum
        and maximum magnitude values for the query. If colNames is not
        NULL, it should be an array of column names to fetch,
        otherwise all columns are fetched. The number of object rows
        returned is limited to maxRows (the acMore() routine can be
        called to see if more objects would have been available).
        On return, numFound is set to the number of objects found and,
        if any were found, "result" is set to point to an AcResult
        object for accessing the catalog values found. The memory for
        the result is allocated (see MEMORY MANAGEMENT above).
        If filename is not NULL, the results will be copied to the
        given file in the format of a local catalog.

        Arguments:
                handle    (in ) - Catalog handle.
                numCols   (in ) - Number of column names specified.
                colNames  (in ) - Array[numCols] of column names to
                                  read, or NULL to read all.
                ra        (in ) - World coordinates of center point
                dec       (in ) - in degrees.
                radius0   (in ) - Min radius from center point.
                radius1   (in ) - Max radius from center point.
                mag0      (in ) - Min magnitude.
                mag1      (in ) - Max magnitude.
                maxRows   (in ) - Max number of rows to return.
                filename  (in ) - If not null, write results to this
                                  file.
                numFound  (out) - Number of objects found.
                result (in/out) - Handle for accessing query results.
        Return value:
                If successful, 0, otherwise 1.


acSearchClosestStar
        Get the values for the specified columns for the object in the
        catalog closest to the given position and return 0 if there
        are no errors. If colNames is not NULL, it should be an array
        of column names to fetch, otherwise all columns are
        fetched. If mag0 and mag1 are not 0.0, they are taken to be
        the minimum and maximum magnitude values for the query.
        "result" is set to point to an AcResult object for accessing
        the catalog values found (it may be empty if none were
        found). The memory for the result is allocated (see MEMORY
        MANAGEMENT above).
```

```
        Arguments:
                handle    (in ) - Catalog handle.
                numCols   (in ) - Number of column names specified.
                colNames  (in ) - Array[numCols] of column names to
                                  read, or NULL to read all columns.
                ra        (in ) - World coordinates of point
                dec       (in ) - in degrees.
                mag0      (in ) - Min magnitude.
                mag1      (in ) - Max magnitude.
                result (in/out) - Handle for accessing query results.
        Return value:
                Returns 0 on success, 1 for errors. It is not considered
                an error if no object is found. Use acrNumRows(result)
                to determine if an object was found.

acCatalogSearch
        Get the values for all objects fulfilling the specified
        criteria.  The criteria are given by the array of column names
        to compare and two arrays of the same size containing min and
        max values, in string format, corresponding to the column
        names. Null strings are ignored, so, for example, if there is
        no min value, only the max value will be used.  If colNames is
        not NULL, it should be an array of column names to fetch,
        otherwise all columns are fetched. The number of object rows
        returned is limited to maxRows (the acMore() routine can be
        called to see if more objects would have been available).  On
        return, numFound is set to the number of objects found and, if
        any were found, "result" is set to point to an AcResult object
        for accessing the catalog values found. The memory for the
        result is allocated (see MEMORY MANAGEMENT above).
        If filename is not NULL, the results will be copied to the
        given file in the format of a local catalog.

        Arguments:
                handle    (in ) - Catalog handle.
                numCols   (in ) - Number of column names specified.
                colNames  (in ) - Array[numCols]  of column  names to
                                  read, or NULL to read all.
                numSearchCols (in ) - number of column names to compare
                searchCols (in ) - array of column names to compare
                minVals   (in ) - Array of column minimum values.
                maxVals   (in ) - Array of column maximum values.
                maxRows   (in ) - Max number of rows to return.
                filename  (in ) - If not null, write results to this
                                  file.
                numFound  (out) - Number of objects found.
                result (in/out) - Handle for accessing query results.
        Return value:
                If successful, 0, otherwise 1.

acMore
        Return true if there would have been more rows available to
        the most recent query, but they were not returned because the
        maximum row (maxRows argument) limit was reached.

        Arguments:
                handle    (in ) - catalog handle
        Return value:
                1 if more rows were available, otherwise 0.

acNumCols
        Return the number of columns in the catalog.
```

```
        Arguments:
                handle    (in ) - catalog handle
        Return value:
                the number of columns in the catalog

acColName
        Return the name of the given column in the catalog, or NULL if
        there is no such column.

        Arguments:
                handle    (in ) - catalog handle
                col       (in ) - column index (0..n)
        Return value:
                column name of NULL if none

acColIndex
        Return the column index for the given column name, or -1 if
        there is no such column.

        Arguments:
                handle    (in ) - catalog handle
                name      (in ) - column name
        Return value:
                column index

acGetError
        Return the text of the most recent error message reported by
        the catalog routines.

        Arguments:
                none
        Return value:
                pointer to error text (a static buffer)

acGetErrorCode
        Return the error code from the most recent error message. This is
        one of the codes defined in <sys/errno.h>, for example: EINVAL for
        an invalid argument value.

        Arguments:
                none
        Return value:
                error code value
```

ACCESSING CATALOG QUERY RESULTS

```
The following routines operate on the AcResult object returned from
the query routines:

acrNumRows
        Return the number of result rows.

        Arguments:
                handle    (in ) - query result handle
        Return value:
                number of rows in result

acrNumCols
        Return the number of result columns.

        Arguments:
```

```
                handle    (in ) - query result handle
        Return value:
                number of columns in result

acrColNames
        Return a pointer to an array of result column names.

        Arguments:
                handle    (in ) - query result handle
        Return value:
                array of column names

acrGetString
acrGetInt
acrGetDouble
acrGetFloat
acrGetShort
acrGetChar
        Get result values by row and column index.
        Note: there are various versions for different data types and
        parameters.  All return 0 for success and set the last
        argument value or return 1 for error.

        Arguments:
                handle    (in ) - query result handle
                row       (in ) - row index
                col       (in ) - column index
                value  (in/out) - address of variable to hold value
        Return value:
                0 for success, otherwise 1 for error.

acrGetNString
acrGetNInt
acrGetNDouble
acrGetNFloat
acrGetNShort
acrGetNChar
        Get result values by row and column name.
        Note: there are various versions for different data types and
        parameters.  All return 0 for success and set the last
        argument value or return 1 for error.

        Arguments:
                handle    (in ) - query result handle
                row       (in ) - row index
                col       (in ) - column name
                value  (in/out) - address of variable to hold value
        Return value:
                0 for success, otherwise 1 for error.

acrGetWC
        Get the position columns (RA and DEC) from the given result
        row and return success (0), otherwise error (1).

        Arguments:
                handle    (in ) - query result handle
                row       (in ) - row index
                pos    (in/out) - address of WC object to hold coords
        Return value:
                0 for success, otherwise 1 for error.

acrColIndex
        Return the result column index for the given result column
```

```
        name.

        Arguments:
                handle    (in ) - query result handle
                colName   (in ) - name of column
        Return value:
                column index or -1 for error.

acrDelete
        Delete the result object (free the memory).
```

SEE ALSO

```
AstroCatalog(3), TclAstroCat(3)
```

\- - - - - -

Last change: 02 Feb 98

4.0.4 AstroImage(n)

NAME

    AstroImage - user interface class for accessing image servers

NAMESPACE

    cat

PARENT CLASS

    util::TopLevelWidget

SYNOPSIS

    AstroImage <path> ?options?

DESCRIPTION

    AstroImage is a user interface class for accessing image servers such
    as DSS (Digitized Sky Survey).

OPTIONS

```
-anchor
        Set the anchor for labels.

-debug
        Flag: if true, run queries in foreground for better debugging.

-image
         image handle (itcl class).

-imagesvr
        Default image server name .

-labelfont
        Font to use for labels.

-labelwidth
        Set the width for  displaying labels.

-title
        Text for  label displayed at top.

-tmpfile
        Temp image file to use.

-valuefont
        Font to use for values.

-valuewidth
        Set the width for  displaying labels.

-wcsfont
        Font to use for ra,dec labels.
```

METHODS

```
add_menubar {}
        Add the menu bar.
```

```
feedback {}
        This method is called by the fileevent handler during the image
        transfer from the HTTP C++ class to give the user feedback about
        how much there is left to copy, etc...  Read a line from the
        feedback pipe, which should contain text to be displayed in the
        progress widget.

fill_imagesvr_menu {m {serv_type "imagesvr"}}
        Fill up a menu of known data servers to choose from.  The
        serv_type argument should be one of: "catalog", "archive",
        "imagesvr".

get_name_servers {m}
        Add the name server catalogs to the given menu.

getimage {args}
        Get the requested image from the image server based on the current
        options and display the resulting image (args are ignored here).

getimage_from_args {ra dec name equinox width height {flag 1}}
        Get the requested image from the image server based on the given
        arguments for the world coord position or name and the given width
        and height if flag is 1, the entries are updated from the
        arguments.

interrupt {}
        Interrupt the current request .

layout_dialog {}
        Do the dialog window layout.

make_short_help {}
        Add a short help window and set the help texts.

quit {}
        Quit and close the window.

request_done {status result}
        This method is called when the background request is done.  The
        arguments are the error status (0 is OK) and the result, which is
        a list of {filename center-pos}.

select_imagesvr {name}
        Called when an imagesvr ($name) is selected from the menu.

set_default_values {}
        Set the default values for the form entries:  set the default
        position (RA, DEC) to the image center, .

set_imagesvr {name}
        Set (open) the current image server - called when user selects a
        name from the menu.

set_namesvr {name}
        Set the name server to use to resolve object names.

set_state {state}
        Set/reset widget states while busy .
```

PROCS

```
instances {}
        Return a list of the instances of this class.
```

```
new_imagesvr {name image debug}
        Member proc (like a static member function) to create an instance
        of this class for the named imagesvr (or reuse the existing one
        for the imagesvr)

        The args are:

          name  - long name of imagesvr from config file

          image - name of skycat itcl image widget

          debug - flag: if true, run queries in foreground.
```

PROTECTED VARIABLES

```
canvas_
         canvas window containing main image.

initialized_
        Flag: set at end of constructor.

logfile_
        Log file handle (used to log URLs).

state_
        Current state: normal, disabled (i.e.: waiting).
```

COMMON CLASS VARIABLES

```
count_
        Count used for filename generation.

instances_
        Array mapping catalog name to widget/class name.

n_instances_
        Instance count.
```

SEE ALSO

```
TopLevelWidget(n)
```

\- - - - - -
Last change: 02 Feb 98

4.0.5 astroImage(3)

NAME

astroImage - C Library for accessing astronomical image servers

SYNOPSIS

```
#include "astroImage.h"

typedef void* AiHandle;

AiHandle aiOpen(char* name);

void aiClose(AiHandle);

char* aiGetImage(AiHandle handle, double ra, double dec,
                 double width, double height);
```

DESCRIPTION

This man page describes a library for accessing astronomical image servers from C applications in a generic way.

Note: Although the library is implemented in C++, it also defines a C interface, which is described here. To use this library from C, an application must have a C++ main and be linked with the C++ compiler. This is easily done by renaming the C main to c_main and defining a C++ main that calls it.

The main purpose of this library is to provide generic access to a variety of astronomical image servers available locally or on the network. The interface hides the details of accessing the various servers.

IMAGE SERVERS

This class communicates with an image server via HTTP requests. The list of servers and how to access them is kept in a configuration file, which is also accessed via HTTP. The default config file is:

    http://archive.eso.org/skycat/skycat.cfg

The default can be overridden by defining the environment variable SKYCAT_CONFIG to another valid HTTP URL.

ERROR HANDLING

Most of the routines described here return non-zero on error and zero if all is OK (routines that return pointers return a NULL pointer on error). The text of the error message can be retrieved with the acGetError() function and an error code is available via the acGetErrorCode function. The error codes are the same as those in <errno.h> and have the same meanings (EINVAL means invalid operand - or parameter, 0 means no code was available).

UNITS

Unless otherwise stated, the units for all radius values are in arcmin. Floating point values for RA and DEC are always in degrees. The default equinox is J2000.

C ROUTINES

```
aiOpen
        Open the named image server and return a handle for it or NULL
        if there were errors. The handle can be used in subsequent
        operations on the image server.
        Arguments:
                name (in) - image server name
        Return value:
                image server handle for use in future calls

aiClose
        Close the image server connection and free its resources.
        Arguments:
                handle (in) - image server handle
        Return value:
                None

aiGetImage
        Pass a request to the image server and return the name of a
        FITS file containing the resulting image, or NULL if not
        found. The return filename is the name of a temporary file
        that will be reused on the next call to this routine.
        Arguments:
                handle (in) - image server handle
                ra        (in ) - world coordinates of center point
                dec       (in ) - in degrees
                width     (in ) - width of image in arc-minutes
                image     (in ) - height of image in arc-minutes
        Return value:
                pointer to temporary FITS file containing image
```

SEE ALSO

AstroImage(3), TclAstroImage(3)

\- - - - - -

Last change: 02 Feb 98

4.0.6 AstroImage3C(3)

NAME

AstroImage3C++ - C++ class for accessing astromical image servers

SYNOPSIS

```
#include "AstroImage.h"

class AstroImage {
...
public:
    AstroImage(const AstroImage&);
    virtual ~AstroImage();

    static AstroImage* open(const char* name);

    int getImage(const WorldCoords& pos, double width, double height);
    int getImage(const char* url);

    static const CatalogInfoEntry* firstCatalog();

    void feedback(FILE* f);

    int status();

    void tmpfile(const char* name);
    const char* tmpfile();

    const char* name();
    const char* longName();
    const char* shortName();
};
```

DESCRIPTION

This class is used to retrieve images from an HTTP based image server based on a name or position and a width and height in arcmin. The main entry point is the "open" method, which returns a pointer to an AstroImage object allocated for use with the named image server.

IMAGE SERVERS

This class communicates with an image server via HTTP requests. The list of servers and how to access them is kept in a configuration file, which is also accessed via HTTP. The default config file is:

    http://archive.eso.org/skycat/skycat.cfg

The default can be overridden by defining the environment variable SKYCAT_CONFIG to another valid HTTP URL.

UNITS

Unless otherwise stated, the units for all radius values are in arcmin. Floating point values for RA and DEC are always in degrees. The default equinox is J2000.

PUBLIC METHODS

open(name)

```
        Open the named image server and return a pointer to an AstroImage
        object allocated for it, or NULL if errors occur.

        Arguments:
                name (in ) - Image server name (long or short name from
                             catalog config file).
        Return value:
                Pointer to an AstroImage object created for the given
                image server.

getImage(pos, width, height)
        Pass a request to the image server to get a FITS file at the
        given position with the given size and return the status.
        The name of the file holding the result can be accessed as
        this->tmpfile().

        Arguments:
                pos    (in ) - World coordinates of center of image.
                width  (in ) - Image width in arcmin.
                height (in ) - Image height in arcmin.
        Return value:
                Status: 0 if OK, 1 for error.

getImage(url)
        Given a URL for the image, request the image from the image
        server and return the status.
        The name of the file holding the result can be accessed as
        this->tmpfile().

        Arguments:
                url    (in ) - HTTP URL for image.
        Return value:
                Status: 0 if OK, 1 for error.


firstCatalog()
        Return a pointer to the first catalog config entry (for link
        list traversal).

feedback(FILE)
        Set the file pointer to use for HTTP feedback during image
        transfers.

status()
        Return the status (after constructor) for error checking.
        A return value of 0 is normal, 1 for errors.

name()
longName()
shortName()
        Return the name (long name) or short name of the image server
        from the catalog config file.

tmpfile()
tmpfile(name)
        Get (or set) the name of the temporary file to use to hold images
        retrieved from the image server.
```

SEE ALSO

AstroCatalog(3), WorldCoords(3)

```
- - - - - -
Last change: 02 Feb 98
```

4.0.7 AstroQuery(n)

NAME

```
AstroQuery - widget for searching astronomical catalogs.
```

NAMESPACE

```
cat
```

PARENT CLASS

```
util::FrameWidget
```

SYNOPSIS

```
AstroQuery <path> ?options?
```

DESCRIPTION

```
An AstroQuery widget is defined as a frame containing entries for
search options.
```

OPTIONS

```
-anchor
        Set the anchor for labels.

-astrocat
        Astrocat (C++ based) object for accessing catalogs.

-command
        Specify the command to evaluate with the results of a query.  The
        command is evaluated whenever a query is done with the following
        args:

          errmsg - If this is not empty, it contains an error message and
          the following args should be ignored. If there were no errors,
          this arg is an empty string.

          headings - are the column headings

          info - is a list of rows (result of query)

          more - is a flag set to 1 if there were more rows available that
          were not returned due to the maxrows limit set.

-debug
        Flag: if true, run queries in foreground for better debugging.

-labelfont
        Font used for labels.

-labelwidth
        Set the width for  displaying labels.

-searchcommand
        Specify the search command to evaluate when the user types <Enter>
        (Can be used to set button states, etc. Defaults to [code $this
        search]).

-valuefont
        Font used for values.
```

```
-wcsfont
        Font used for ra,dec labels.
```

METHODS

```
add_copyright {}
        Add copyright info to the search panel, if present.

add_search_option {args}
        Add a row to the search options panel (blt table).  The args may
        be one or more widget names to put in a row in the panel.  Each
        widget is placed in a new column. An empty argument results in an
        empty column for that row. If all are empty, the row is left
        empty.

add_search_options {}
        Add (or update) the search options panel.

do_query {cmd}
        Return the result of a catalog query. Since this may be run in a
        separate process via fork, all of the necessary info is returned
        in the format:

         {results headings entry querypos more}

        where:

         results are the query results (list of rows)

         headings are the result headings

         entry is the catalog config entry

         querypos is the center pos of the query

         more is a flag, 1 if more rows available.

get_catalog_equinox {}
        Return the current equinox, or an empty string if the catalog does
        not support world coordinates. The user can change the equinox
        used to display coordinates by typing in a different value in the
        panel.

get_pos_radius {}
        Get the values from the position and radius entries and return a
        list in the format {ra dec equinox radius} if we are using wcs, or
        {x y radius} for image pixel coords.

init {}
        Called after options have been evaluated.

interrupt {}
        Interrupt the current search .

query_done {status result}
        This method is called when the background query is done.  The
        arguments are the error status (0 is OK) and the result of
        evaluating the Tcl commands.

        The result arg contains a list of items:

          {info headings entry pos more}
```

```
        Where:

          info is a list of rows (result of query)

          headings are the column headings

          entry is the catalog config entry (including info from the
          header in the query result),

          pos is the query center position (possibly expanded by a name
          server)

          more is a flag indicating if there were more rows available that
          were not returned due to the maxrows limit set.

search {args}
        Start the catalog search based on the current search options and
        display the results in the table.

set_default_values {}
        Set the default values for the form entries:  set the default
        position (RA, DEC) to the image center, set the min radius to 0
        and the max radius to the distance from the center to the origin.

set_pos_radius {list}
        Set the values for the position and radius entries from the given
        list, which should be in the format {ra dec equinox radius} if we
        are using wcs, or {x y radius} for image pixel coords.

set_short_help {}
        Add short help texts to widgets.

update_search_options {}
        Update the search option entries after they have been edited.
```

PROTECTED VARIABLES

```
min_values_
        Array(colName) of widget name for search column min fields.

name_
        Widget name for name field.

search_col_info_
        Saved value of catalog search_cols_ entry, for comparison.

search_opts_
        Blt table for search options.
```

SEE ALSO

```
FrameWidget(n)
```

```
- - - - - -
Last change: 02 Feb 98
```

4.0.8 AstroQuery3C(3)

NAME

AstroQuery3C++ - A class describing a query to search an astronomical catalog

SYNOPSIS

```
#include "AstroQuery.h"

class AstroQuery {
...
public:
    AstroQuery();
    AstroQuery(const AstroQuery&);
    ~AstroQuery();


    const char* id() const;
    int id(const char* s);

    const WorldCoords& pos() const;
    int pos(const WorldCoords& p);

    int pos(const WorldCoords& p1, const WorldCoords& p2);

    double width() const;
    double height() const;
    int dim(double w, double h);

    double mag1() const;
    double mag2() const;
    int mag(double m);
    int mag(double m1, double m2);

    double radius1() const;
    double radius2() const;
    int radius(double r);
    int radius(double r1, double r2);

    char** colNames() const;
    const char* colName(int col) const;
    int numCols() const;
    int colNames(int n, char** ar);

    const char* sort() const;
    int sort(const char* s);

    int maxRows() const;
    int maxRows(int n);

    int numSearchCols() const;
    char** searchCols() const;
    char** minValues() const;
    char** maxValues() const;
    int condition(int numSearchCols, char**searchCols,
                  char**minVals, char**maxVals);
};
```

DESCRIPTION

This class is used in star catalog queries to specify which object(s) to search for. The class attributes specify the conditions for the search, such as id, name, position, radius, etc. All of the fields are optional. Fields are set to the appropriate null value, if they are not being used.

PUBLIC METHODS

All of the public methods in this class simply access class members (set and get member values). Methods with arguments set the member values and return the error status. Methods with no arguments return the value.

```
id() const
id(const char* s)
        Get or set the value for the object id to search for.

pos() const
pos(const WorldCoords& p)
        Get or set the center position for a radius search.

pos(const WorldCoords& p1, const WorldCoords& p2)
        Calculate center position, width and height from the 2 corner
        positions.

width() const
height() const
        Get the value of the width or height fields (in arcmin).

dim(double w, double h)
        Set the values of the width and height fields in arcmin.

mag1()
mag2()
        Get the max (or min and max) magnitude fields.

mag(double m);
        Set the max magnitude for the query (min is 0.0).

mag(double m1, double m2)
        Set min/max magnitude with check.

radius1() const
radius2() const
        Get max (or min and max) radius values in arcmin.

radius(double r)
        Set the max radius for the query in arcmin (min is 0.0).

radius(double r1, double r2)
        Set min/max radius with check.

colNames() const
        Return array of column names.

colName(int col) const
        Return name of given column.

numCols() const
        Return number of columns.

colNames(int n, char** ar)
        Set the number of columns and the column names.
```

```
sort() const
sort(const char* s)
        Get or set the sort column.

maxRows() const
maxRows(int n)
        Get or set the max number of rows to return from a query.

numSearchCols() const
searchCols() const
minValues() const
maxValues() const
condition(int numSearchCols, char**searchCols, char**minVals,
        char**maxVals)
        Get or set the search conditions as min and max values for
        given columns (Not implemented yet).
```

SEE ALSO

AstroCatalog(3)

\- - - - - -

Last change: 02 Feb 98

4.0.9 CatalogInfo(n)

NAME

CatalogInfo - widget for browsing through a hierarchical list of catalogs

NAMESPACE

cat

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

CatalogInfo <path> ?options?

DESCRIPTION

The CatalogInfo class defines a user interface for browsing through a hierarchical list of catalogs. The list of catalogs is defined in catalog config files at various hosts on the network or in local catalog config files. This widget allows you to choose, from various catalog servers on the net, which catalogs to keep in a personal default catalog list.

OPTIONS

-anchor
: Set the anchor for labels.

-classname
: Name of a subclass of AstroCat to use to create new catalog widgets.

-command
: Command to eval when done.

-debug
: Debugging flag.

-id
: Optional unique id, used in searching for already existing catalog widgets.

-labelfont
: Font to use for labels.

-labelwidth
: Set the width for displaying labels.

-valuefont
: Font to use for values.

METHODS

add_catalog {}
: Add the selected catalog to the catalog list.

add_dialog_buttons {}
: Add the dialog button frame.

```
add_menubar {}
        Add the menu bar.

add_node {path name {before ""}}
        Add a node to the catalog tree. "path" is the path to the parent
        of the new node in the format "a${sep_}b${sep_}c" where a is the
        root and c is immediate parent of the new node. "name" is the name
        of the new node and the text to display. If "before" is specified,
        the node is inserted just before the node named "$before".

add_tree_frame {}
        Add a frame for displaying a tree of catalog servers and catalogs
        inside the given frame and side.

apply {{ask 0}}
        Apply the current changes, save to default file, call $command.

close {{ask 0}}
        Apply changes and close the window .

close_tree_node {path}
        Called when a tree node is closed .

init {}
        Called after options have been evaluated.

init_tree {}
        Initialize the tree of catalogs.

insert_node {name}
        Insert the given catalog in the tree under the root, if it is no
        already there, and keep the tree sorted.

layout_dialog {}
        Do the dialog window layout.

make_short_help {}
        Add a short help window and set the help texts.

open_catalog {args}
        Open a window to search the selected catalog.

open_config_file {}
        Pop up a dialog to open and load a catalog config file.

open_tree_node {path}
        Called when a tree node is opened .

reinit_tree {}
        Reinitialize the tree.

remove_catalog {}
        Remove the selected catalog from the catalog list.

select_catalog {path}
        Called when a catalog is selected in the catalog list.

toggle_tree_layout {}
        Toggle the layout of the tree.
```

PROCS

```
save {file w {ask 1}}
```

```
        Save current config info to the given file, or if file is an empty
        string, to the default config file.

save_as {w}
        Pop up a dialog to save the current info to a catalog config file
        Note: this is defined as a proc so that it can be accessed by
        other widgets without this widget being created first. $w should
        be the main window of the calling widget.
```

PROTECTED VARIABLES

```
sep_
        Separator char for Tix tree/hlist widget.

tree_
        Tix tree widget.
```

COMMON CLASS VARIABLES

```
astrocat_
        C++ astrocat object used by static member procs.
```

SEE ALSO

```
TopLevelWidget(n)
```

```
- - - - - -
Last change: 02 Feb 98
```

4.0.10 CatalogInfo3C(3)

NAME

CatalogInfo3C++ - C++ class managing access to catalog configuration files

SYNOPSIS

```
#include "CatalogInfo.h"

class CatalogInfo {
...
public:
    CatalogInfo();

    static int load();
    static int reload();
    static CatalogInfoEntry* load(istream&, const char* filename = "internal");
    static int load(CatalogInfoEntry*);
    static CatalogInfoEntry* lookup(const char* catalogName);
    static CatalogInfoEntry* lookup(CatalogInfoEntry* entry, const char* name);
    static CatalogInfoEntry* lookupFile(const char* catalogFileName);
    static void updateConfigEntry(istream& is, CatalogInfoEntry* entry);
    static int append(CatalogInfoEntry* e);
    static int remove(CatalogInfoEntry* e);
    static CatalogInfoEntry* first();
    static CatalogInfoEntry* root();
};
```

DESCRIPTION

This class is used to load, edit, access and save catalog config files either locally or via HTTP from a remote host. Config files are referenced by a URL. This may be either http:/host/file for remote config files or file:/pathname for local files. This is a static class, since all of the information is meant to be globally available. The most common method called is "lookup", which returns a pointer to the catalog entry for a given catalog name.

The default config file is loaded automatically the first time it is needed. There is a hard coded default URL that points to ESO's master config file. The environment variable CATLIB_CONFIG

METHODS

CatalogInfo()

Constructor. Note that this is a static class (all methods are static), so no instances are required.

load()

Load the default catalog config file (called automatically).

reload()

Reload the default catalog config file after it has been edited by hand.

load(istream, filename)

Load a config file info from the given stream (filename is used for error reporting).

load(entry)

Load the catalog config file given by the entry's URL field.

```
        This is used for directory entries, to follow a catalog
        directory link.

lookup(catalogName)
        Return a pointer to the catalog config file entry for the
        given catalog.

lookup(entry, name)
        Look up the catalog name as above, but start the search
        with the given directory entry rather than at the root.

lookupFile(catalogFileName)
        Get the config entry for a local catalog from the header

updateConfigEntry(istream, entry)
        Read catalog config keyword entries from the given stream and
        update the given entry values.

append(entry)
        Append the given entry to the end of the main catalog list.

remove(entry)
        Remove the given entry from the main catalog list.

first()
        Return a pointer to the first config file entry under the root
        entry.

root()
        Return a pointer to the root config file entry.
```

SEE ALSO

```
CatalogInfoEntry(3), AstroCatalog(3),
Catalog Library (Overview, part of Frame doc)
```

\- - - - - -

Last change: 02 Feb 98

4.0.11 CatalogInfoEntry(3)

NAME

CatalogInfoEntry - C++ class representing a catalog config file entry

SYNOPSIS

```
#include "CatalogInfo.h"

class CatalogInfoEntry {
...
public:
    CatalogInfoEntry();
    CatalogInfoEntry(const CatalogInfoEntry&);

    int operator=(const CatalogInfoEntry&);

    ~CatalogInfoEntry();

    char* check();

    void servType(const char* s);
    void longName(const char* s);
    void shortName(const char* s);
    void url(const char* s);
    void backup1(const char* s);
    void backup2(const char* s)
    void symbol(const char* s);
    void searchCols(const char* s);
    void sortCols(const char* s);
    void sortOrder(const char* s);
    void showCols(const char* s);
    void copyright(const char* s);
    void id_col(int i);
    void ra_col(int i);
    void dec_col(int i);
    void x_col(int i);
    void y_col(int i);
    void is_tcs(int i);
    void equinox(double d);

    const char* servType() const;
    const char* longName() const;
    const char* shortName() const;
    const char* url() const;
    const char* backup1() const;
    const char* backup2() const
    const char* symbol() const;
    const char* searchCols() const;
    const char* sortCols() const;
    const char* sortOrder() const;
    const char* showCols() const;
    const char* copyright() const;
    int id_col() const;
    int ra_col() const;
    int dec_col() const;
    int x_col() const;
    int y_col() const;
    int is_tcs() const;
    double equinox() const;

    int isWcs();
```

```
    CatalogInfoEntry* link() const;
    void link(CatalogInfoEntry*e);

    int append(CatalogInfoEntry* e);

    CatalogInfoEntry* next() const;
    void next(CatalogInfoEntry*e);

    friend ostream& operator<<(ostream&, const CatalogInfoEntry&);
};
```

DESCRIPTION

```
Objects of class CatalogInfoEntry are used to represent catalog config
file entries. The entries are linked together in the form of a
hierarchical list.  The next pointer points to the next catalog in the
list and the link pointer points to the next list in the hierarchy.

A config file entry contains the information necessary to access a
catalog. The syntax for each catalog entry is:

  serv_type:   service type, one of: catalog, namesvr, imagesvr
               directory, local... (see SERVICE TYPES below)

  long_name:   long name of service for displaying
  short_name:  short name of service
  url:         URL used to access catalog, %ra,%dec, etc. expanded
               (see below)

  symbol:      the symbol to use to plot the given column value
               (see SYMBOLS below)

  copyright:   optional copyright notice to display in user interface

  search_cols: optional list of columns that can be searched by in the format
               col1 "col1 min label" "col1 max label" : ...
               example:
                  search_cols:    mag "Brightest (min)" "Faintest (max)"

  sort_cols:   optional list of columns to sort by {col1 col2 ...}
  sort_order:  optional: set to "increasing" or "decreasing"

  show_cols:   optional list of columns to display (default: all)

  id_col:      column containing id field
  ra_col:
  dec_col:     columns containing ra and dec (for catalogs supporting WCS)

  x_col:
  y_col:       columns containing x,y coords (for catalogs not supporting WCS)

  is_tcs:      flag: true if using TCS columns
```

SERVICE TYPES

```
The currently known service types are:

  catalog  - server returns a tab separated table of row/col values

  namesvr  - server returns a single line with id, ra and dec to resolve
             the given object name
```

```
  imagesvr - server returns an image file

  directory - the URL is a pointer to another catalog config file

  local - a local catalog
```

URL SYNTAX

```
The url field is used to build a URL to get the results via HTTP.
The syntax is like this:

      http://host:port/cgi-bin/server?arg1&arg2&...argn

If ":port" is missing, it defaults to 80.

Substitutions are performed on the URL as follows:

  %ra, %dec        - coordinates of center point

  %w, %h           - width and height in arcmin

  %r1, %r2         - min and max radius (for circular query)
  %r               - use when server only accepts single radius value

  %m1, %m2         - min and max magnitude
  %m               - use when server only accepts single magnitude

  %n               - max number of rows to return

  %cols            - list of columns to return (col1,col2,...coln)

  %id              - ID field of item to return (if supported)

  %mime-type       - value for http mime-type field

Name servers only need the %id field, which is set to the object name.
```

SYMBOLS

```
The syntax for the "symbol:" field is as follows:

    symbol:  colnames symbol expr : colnames symbol expr : ...

 where
   colnames - is a list of column names used (in symbol or expr)

   symbol - is the symbol to use, one of: square, circle, triangle,
            cross, plus, diamond, ellipse.  The symbol may also be a
            list such as {circle yellow} and some symbols take extra
            args for ratio and angle (ellipse).

   expr - is an expression in terms of colnames above, used to
          determine the size of the symbol. It may also be a list
          {expr units}, where units is one of {"image" "deg J2000"
          "deg B1950}. The default is "image" pixel coordinates.  See
          the RTD documentation for details on the syntax of
          coordinates and units. The column names (colnames) can be
          used as variables in the expression using "$" (following Tcl
          syntax).

 example:
    symbol: mag circle 15-$mag : xyz square (1-$xyz)*2.5
    symbol: {a/b pa mag} {ellipse yellow ${a/b} $pa} {15-$mag}
```

```
symbol: "a/b pa mag" "ellipse yellow ${a/b} $pa" "15-$mag"
```

METHODS

```
CatalogInfoEntry()
        Default constructor, sets all fields to NULL or default
        values. Use methods to set individual values.

CatalogInfoEntry(entry)
        Copy constructor.

operator=(entry)
        Assignment operator.

~CatalogInfoEntry()
        Destructor.

check()
        Check that all required fields have been set and return 0 if
        all ok.

servType(s)
longName(s)
shortName(s)
url(s)
backup1(s)
backup2(s)
symbol(s)
searchCols(s)
sortCols(s)
sortOrder(s)
showCols(s)
copyright(s)
        Set string keyword values (copy is made internally).

id_col(i)
ra_col(i)
dec_col(i)
x_col(i)
y_col(i)
is_tcs(i)
        Set int keyword values.

equinox(d)
        Set the equinox for the entry.

servType()
longName()
shortName()
url()
backup1()
backup2()
symbol()
searchCols()
sortCols()
sortOrder()
showCols()
copyright()
        Get string keyword values.

id_col()
ra_col()
dec_col()
```

```
x_col()
y_col()
is_tcs()
        Get int keyword values.

equinox()
        Get the equinox.

isWcs()
        Return true if the catalog uses word coordinates, that is, if
        ra_col and dec_col have valid values The default is to use
        World Coordinates with ra_col=1, dec_col=2.

link()
link(entry)
        Get or set the pointer to link entry. Link entries are for
        catalog directory entries, where the link points to the
        beginning of a new sublist of catalogs.

append(entry)
        Append the given entry to the end of this list.

next()
next(entry)
        Get or set the pointer to next entry in the list.

operator<<(ostream, entry)
        Output operator. The output format is like the config file
        format.
```

SEE ALSO

CatalogInfo(3), AstroCatalog(3)

\- - - - - -

Last change: 02 Feb 98

4.0.12 EnterObject(n)

NAME

    EnterObject - Widget for entering data to add to a local catalog.

NAMESPACE

    util

PARENT CLASS

    util::EntryForm

SYNOPSIS

    EnterObject <path> ?options?

DESCRIPTION

    EnterObject is a dialog widget for entering object data to add to a
    local catalog.

OPTIONS

    -image
            Optional image handle (itcl class RtdImage or derived class).

METHODS

    cancel {}
            Called for the Cancel button.

    init {}
            Called after options have been evaluated.

    pick_object {}
            This method is called to pop up a dialog to allow the user to pick
            an object in the image.

    picked_object {list}
            This is called when an object has been selected with the pick
            dialog.

SEE ALSO

    EntryForm(n)

\- - - - - -
Last change: 02 Feb 98

4.0.13 LocalCatalog(3)

NAME

LocalCatalog - Class for accessing local catalogs stored as tab tables

SYNOPSIS

```
#include "LocalCatalog.h"

class LocalCatalog : public AstroCatalog {
...
public:
    LocalCatalog(const CatalogInfoEntry*);

     virtual int query(const AstroQuery& q, const char* filename, QueryResult& re-
sult);

    static int check_table(const char* file);

    virtual int sort(const char* column, int numeric = 1);

    virtual int uniq();
};
```

DESCRIPTION

```
This class can be used to search a local catalog stored as a file in
tab table format. The format of the table is something like:

TableName

# comment line

# catalog config information
keyword: value ...
```

A B C

```
-       -       -
0       1       3
3       2       4
...
```

where the table name and catalog config info are optional. This is the same format used by the "Starbase" package or utilities (see: http://cfa-www.harvard.edu/~john/ for more about Starbase).

UNITS

Unless otherwise stated, the units for all radius values are in arcmin or image pixels, depending on the catalog config entry settings. The default is arcmin. Floating point values for RA and DEC are always in degrees. The default equinox is J2000, unless set otherwise in the catalog config entry or in the table header.

PUBLIC METHODS

LocalCatalog(const CatalogInfoEntry*)

Constructor - create local catalog class instance.
Note: public interface normally uses AstroCatalog::open() with the name of the file containing the local catalog. The

```
        argument represents the entry in the catalog config file for
        this catalog (made automatially, if not already present).

query(const AstroQuery& q, const char* filename, QueryResult& result)
        Run a query on the catalog and return the number of objects
        found (redefined here to work with local catalogs, refer to
        the parent class AstroCatalog(3) for details.).

check_table(const char* file)
        Check the validity of a tab table file and return 0 if it is ok.
```

SEE ALSO

AstroCatalog(3)

\- - - - - -

Last change: 02 Feb 98

4.0.14 PreviewPlot(n)

NAME

PreviewPlot - Widget for displaying a graph of tab table data

NAMESPACE

cat

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

PreviewPlot <path> ?options?

DESCRIPTION

PreviewPlot is an itcl widget for displaying a graph of tab table data, such as that returned from a "Preview" URL in catalogs such as the HST.

OPTIONS

-file
    File containing preview data in tab table format.

-name
    Name of object/star (for title).

METHODS

dispXY {x y}
    Display x, y values at cursor position.

make_buttons {}
    Make the button frame at the bottom of the window.

make_graph {}
    Make the graph subwindow.

make_short_help {}
    Add a short help window and set the help texts.

plot {file}
    Plot the data in the file .

print {}
    Make a hard copy of the graph display.

quit {}
    Quit the window.

PROTECTED VARIABLES

graph_
    Name of graph widget.

SEE ALSO

TopLevelWidget(n)

```
- - - - - -
Last change: 02 Feb 98
```

4.0.15 QueryResult(n)

NAME

QueryResult - Widget for viewing the results of a catalog query.

NAMESPACE

cat

PARENT CLASS

util::TableList

SYNOPSIS

QueryResult <path> ?options?

DESCRIPTION

A QueryResult widget frame is defined as a TableList (see TableList(n) in the tclutil package) with some methods added for catalog support. It is used to display the results of a catalog query. See also AstroQuery(n) for the query paramaters, AstroCat(n) for the main window or classes derived from these in the skycat package. These classes do not deal with images - only catalog data. See the derived classes in the skycat package for image support.

OPTIONS

-astrocat
    Astrocat (C++ based) object for accessing catalogs.

METHODS

add_rows {info headings}
    Add the given info rows (result of a query) to a local catalog file with the given headings. The user selects the name of the catalog file.

add_selected {}
    Add the currently selected rows to a local catalog file.

add_to {}
    Add the rows in the current listing to a local catalog file.

check_row {data}
    Check that the given row contains valid data for a catalog and return 0 if OK.

edit_selected_object {{command ""}}
    Pop up a window so that the user can edit the selected object(s). The optional command is evaluated with no args if the object is changed.

enter_new_object {{command ""}}
    Pop up a dialog to enter the data for a new object for a local catalog. The command is evaluated after the users enters the new data.

enter_object {command info}
    This method is called with the data for a new object to add to a local catalog. The row is added to the local catalog file and the

```
        command is evaluated.

print {}
        Pop up a dialog to print the table listing.

remove_selected {}
        Remove the currently selected rows from a local catalog file.

replace_object {command old_data new_data}
        This method is called with the new data to replace the data for
        the selected object/row in the local catalog. The command is
        evaluated when done.

reset {}
        Reset table dialogs if needed.

save_as {}
        Save the current data to a local catalog.

save_to_file {file info headings {iflag 0}}
        Save the given info (the result of query) to the given catalog
        file, using the given column headings.  If iflag is 1, insert rows
        in the existing file.

select_columns {}
        Pop up a dialog to select table columns to display.

sort_dialog {}
        Pop up a dialog to sort the list.

update_options {}
        Update the table sort and column display options from the catalog
        entry.
```

SEE ALSO

`TableList(n)`

\- - - - - -

Last change: 02 Feb 98

4.0.16 QueryResult3C(3)

NAME

QueryResult3C++ - Class for accessing results of a catalog query

SYNOPSIS

```
#include "QueryResult.h"

class QueryResult : public TabTable {
public:
    QueryResult();
    QueryResult(const char* result);
    QueryResult(int numCols, char** colNames, const char* result);
    virtual ~QueryResult();

    virtual int getPos(int row, WorldCoords& pos);
    virtual int getPos(int row, WorldOrImageCoords& pos);

    virtual int query(const AstroQuery& q, const char* infile, const char* outfile,
              int& more);

    virtual void entry(CatalogInfoEntry* e, const char* result = NULL);

    virtual const char* symbol() const;
    virtual const char* copyright() const;
    virtual int id_col() const;
    virtual int ra_col() const;
    virtual int dec_col() const;
    virtual int x_col() const;
    virtual int y_col() const;
    virtual double equinox() const;

    virtual int isWcs() const;
};
```

DESCRIPTION

This class manages the result of an AstroCatalog::query. The basic result is a char buffer that contains one row per line where each column is separated by a tab character. Since this is the same format as a local catalog, this class is also used to implement local catalogs (see LocalCatalog(3)).

This class provides transparent access to the result based on a row,column index and allows for type conversion from string to the desired type.

This class is the same as class TabTable (the parent class), except that it assumes the table has at least the 3 columns "id", "ra" and "dec", if World Coordinates are supported, or "id", "x" and "y" if image coordinates are being used. This class adds methods to get the position for a row and to search in an circle, given the center position and radius. The class keeps a pointer to a catalog config entry that may be based on the original catalog entry or may be read from the header of the local catalog or query results. World Coordinates are assumed if the entry indicates that ra and dec columns are present.

UNITS

Unless otherwise stated, the units for all radius values are in arcmin

```
for World Coordinates and image pixels otherwise..  Floating point
values for RA and DEC are always in degrees. The default equinox is
J2000.
```

PUBLIC METHODS

```
QueryResult()
QueryResult(const char* result)
QueryResult(int numCols, char** colNames, const char* result)
        The constructors have the same arguments as for the parent
        class TabTable.  They initialize the table from a buffer in
        tab table format. See the TabTable(3) constructors for
        details.

getPos(int row, WorldCoords& pos)
getPos(int row, WorldOrImageCoords& pos)
        Get the world coordinate or image pixel position for the given
        row from the columns "ra" and "dec", or "x" and "y", depending
        on the settings in the catalog config entry. By default the
        first 3 fields are: id, ra, dec, in J2000. Keywords in the
        catalog config entry may override these settings. The return
        value if 0 unless errors are detected.

query(const AstroQuery& q, const char* infile, const char* outfile,
      int& more)

        Query the given tab table file using the condition described
        by the given AstroQuery object.

        Args:
            q -        (in)   object describing the query

            infile   - (in)   file to search
            outfile  - (in)   optional filename to hold results, or null

            more   -   (out)  set to 1 if more objects would be available
                              but were not returned because q.maxRows was
                              set lower

        The return value 0 if all is OK. The number found is available
        as this->numRows().

entry(e, result)
        Set the catalog config entry for this object. This is included
        in the file when this object is saved as a local catalog. The
        optional result arg may be a pointer to the result of a
        catalog query, which may contain config configuration
        information. If specified, it is scanned to update the entry
        with the new information.

symbol()
id_col()
ra_col()
dec_col()
x_col()
y_col()
equinox()
        All these methods return the corresponding values from the
        catalog config entry for the query results. This may be the
        same as for the catalog being searched or may have been
        modified by config information in the header of the query
        results or local catalog.
```

```
isWcs()
        Returns true if the query results contain ra and dec columns
        for World Coordinate support.
```

SEE ALSO

```
AstroCatalog(3)
```

\- - - - - -

Last change: 02 Feb 98

4.0.17 SearchConfig(n)

NAME

SearchConfig - Widget for editing a list of search columns

NAMESPACE

cat

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

SearchConfig <path> ?options?

DESCRIPTION

Class SearchConfig defines dialog window for editing the list of search columns for the AstroCat catalog window. This allows you to specify a list of columns to search by, save the information in a local catalog config file and reconfigure the current windows to include entries for the search columns.

OPTIONS

-anchor
    Set the anchor for labels.

-astrocat
    Astrocat (C++ based) object.

-catalog
    Catalog name.

-command
    Command to eval when done.

-labelfont
    Font to use for labels.

-labelwidth
    Set the width for displaying labels.

-valuefont
    Font to use for values.

METHODS

add_dialog_buttons {}
    Add the dialog button frame.

add_tables {}
    Add a table listing the search columns and entries to edit.

apply {{ask 0}}
    Apply the changes and close the window.

fill_tables {}
    Fill the left and right tables with the column info.

```
get_row {}
        Return a row based on the current entries or empty if not all of
        the entries contain values.

init {}
        Called after options have been evaluated.

make_short_help {}
        Add a short help window and set the help texts.

select_row {tab}
        Called when a table row (left or right) is selected.

set_row {args}
        Update the selected row with the typed in values.

update_selection {}
        Update the selection in the listboxes after a change.
```

PROTECTED VARIABLES

```
left_
        Left listbox widget.

save_needed_
        Set to true if the catalog configuration has been edited and needs
        to be saved.
```

SEE ALSO

```
TopLevelWidget(n)
```

- - - - - -
Last change: 02 Feb 98

4.0.18 SymbolConfig(n)

NAME

SymbolConfig - Widget for editing plot symbol information

NAMESPACE

cat

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

SymbolConfig <path> ?options?

DESCRIPTION

Class SymbolConfig defines a dialog window for editing the plot symbol information for the AstroCat catalog window. The dialog lets you select which symbols to plot under given conditions, the size, color, shape of the symbols, specified as expressions in terms of column name variables.

OPTIONS

-anchor
Set the anchor for labels.

-astrocat
Astrocat (C++ based) object.

-catalog
Catalog name.

-colors
Color list for color menu.

-command
Command to eval when done.

-labelfont
Font to use for labels.

-labelwidth
Set the width for displaying labels.

-valuefont
Font to use for values.

METHODS

add_dialog_buttons {}
Add the dialog button frame.

add_row {}
Add a new row to the table.

add_table {}
Add a table listing the symbols and some widgets to edit them.

```
apply {args}
        Apply the current changes.

cancel {}
        Called when the cancel button is pressed. Delete the window so
        that it will be reinitialized next time.

fill_table {}
        Fill the table with the symbol info for the current catalog.

get_row {}
        Return a table row based on the current entries and menus or empty
        if no data was enterred. Raises an error if there was something
        wrong.

init {}
        Called after options have been evaluated.

make_short_help {}
        Add a short help window and set the help texts.

ok {}
        Apply the changes and close the window.

remove_row {}
        Remove the selected row from the table.

save {{ask 0}}
        Save the changes in the default config file.

select_row {}
        Called when a table row is selected.

set_symbol {symbol}
        Called when a symbol is selcted from the menu.

update_table {args}
        Update the table display from the entries and return an empty
        string if all was OK.
```

PROTECTED VARIABLES

```
save_needed_
        Set to true if the catalog configuration has been edited and needs
        to be saved.

table_
        TableList widget displaying plot info.
```

SEE ALSO

```
TopLevelWidget(n)
```

_ _ _ _ _ _

Last change: 02 Feb 98

4.0.19 TabTable(3)

NAME

TabTable - Class for managing a tab separated table of values

SYNOPSIS

```
#include "TabTable.h"
class TabTable {
...
public:
    // constants
    enum {MAX_ROW_SIZE=8*1024};  // max size of a tab table row
    enum {MAX_HEADER_SIZE=1024}; // max size of a tab table heading
    enum {MAX_COLUMNS=255};      // max number of table columns

public:
    TabTable(char sep = '\t');
    TabTable(const char* buf, int maxRows = 0, char sep = '\t');
    TabTable(int numCols, char** colNames, const char* buf,
             int maxRows = 0, char sep = '\t');
    virtual ~TabTable();

    virtual int clear();

    virtual int init(const char* buf, int maxRows = 0);
    virtual int init(int numCols, char** colNames, const char* buf,
                     int maxRows = 0);

    virtual int get(int row, int col, char*& value);
    virtual int get(int row, int col, int& value);
    virtual int get(int row, int col, double& value);
    virtual int get(int row, int col, float& value);
    virtual int get(int row, int col, short& value);
    virtual int get(int row, int col, char& value);

    virtual int get(int row, const char* colName, char*& value);
    virtual int get(int row, const char* colName, int& value);
    virtual int get(int row, const char* colName, double& value);
    virtual int get(int row, const char* colName, float& value);
    virtual int get(int row, const char* colName, short& value);
    virtual int get(int row, const char* colName, char& value);

    virtual int colIndex(const char* colName)  const;
    virtual const char* colName(int col)  const;
    virtual int hasCol(const char* name)  const;

    static int head(const char* filename, TabTable&);
    static int head(istream&, TabTable&);

    virtual int compareHeadings(const TabTable& t);

    virtual int save(const char* filename);
    virtual int save(ostream&);

    virtual int append(const char* filename);
    virtual int remove(const char* filename);

    virtual int findRow(const char* tableRow);
    virtual int printRows(ostream& os);

    virtual int search(const char* filename, int numSearchCols,
                       char** searchCols,
```

```
                       char** minValues, char** maxValues,
                       int maxRows);

    virtual int search(const char* filename, const char* searchCol,
                       const char* value, int maxRows);

    virtual int search(const char* filename, int searchCol,
                       const char* value, int maxRows);

    friend ostream& operator<<(ostream& os, TabTable& t);

    virtual int numRows() const;
    virtual void numRows(int n);
    virtual int numCols() const;
    virtual char** colNames() const;
    virtual int status() const;
};
```

DESCRIPTION

Class TabTable manages a buffer containing a table where the rows are separated by newlines and the columns by tabs (or other specified char). The class provides transparent access to the table based on a row,column index and allows for type conversion from string to the desired type. Methods are also available to print the table and save it to a file.

PUBLIC METHODS

TabTable(char sep = '\t')

Constructor: initialize empty table.

TabTable(const char* buf, int maxRows = 0, char sep = '\t')

Constructor: initialize table from buffer in tab table format:

```
VAR     VALUE   VALUE   ...
...
COL1    COL2    COL3    ...     COLN
---             ----    ----    ----
data    data    data            data
...
```

Currently, anything up to the column headings is ignored (This may change later). The format is the same as the tab table format used by starbase. A line beginning with a "-" separates the column names from the data and all column headings and data are separated by tabs (or sep char).

The part preceding the column headings may be used later to set certain variables pertaining to the table. In this case, a variable may have one or more values, separated by tabs (sep char).

If maxRows is nonzero, only upto that many rows are taken from buf.

TabTable(int numCols, char** colNames, const char* buf, int maxRows = 0, char sep = '\t')

Constructor: initialize table from data buffer without If maxRows is nonzero, only upto that many rows are taken from buf. headings. The first two args specify the number column headings and their names.

```
~TabTable()
        Destructor: free any allocated memory

init(const char* buf, int maxRows = 0)
        Fill the table from the given buffer in tab table format.  If
        maxRows is nonzero, only upto that many rows are taken from
        buf.

init(int numCols, char** colNames, const char* buf, int maxRows = 0)
        Fill the table from the given buffer in tab table format, with
        headings specified separately.  The first two args specify the
        number column headings and their names.  If maxRows is
        nonzero, only upto that many rows are taken from buf.

clear()
        Make the table empty and free any resources used

get(int row, int col, char*& value)
get(int row, int col, int& value)
get(int row, int col, double& value)
get(int row, int col, float& value)
get(int row, int col, short& value)
get(int row, int col, char& value)
        Get row,column values: the parameter "value" is set and 0 is
        returned if there are no errors. Values of type char* point to
        internal storage and should not be modified.

get(int row, const char* colName, char*& value)
get(int row, const char* colName, int& value)
get(int row, const char* colName, double& value)
get(int row, const char* colName, float& value)
get(int row, const char* colName, short& value)
get(int row, const char* colName, char& value)
        Get table values by column name. Values of type char* point to
        internal storage and should not be modified.

colIndex(const char* colName)  const
        Return the table column index for the given table column name.

const char* colName(int col)  const
        Return the column name for the given column index.

hasCol(const char* name)  const
        Return true if the table contains the given column.

static int head(const char* filename, TabTable&)
static int head(istream&, TabTable&)
        Read the heading info from the given stream and put it in the
        given table.

compareHeadings(const TabTable& t)
        Compare headings in this table and the given one and return 0
        if they are the same.

save(const char* filename)
save(ostream&)
        Save the contents of this object to the given file as a tab
        table.

append(const char* filename)
        Append the contents of this object to the given tab table file
```

```
remove(const char* filename)
        Remove rows in the tab table file that are also in this object.

findRow(const char* tableRow)
        Find a row matching the given tab separated one and return the
        index.

search(const char* filename, int numSearchCols, char** searchCols,
       char** minValues, char** maxValues, int maxRows)

        Search the given tab table file for upto maxRows rows with
        columns values matching the given arguments and fill this
        table with the resulting rows.

        Args:

        filename          - tab table file to search
        numSearchCols     - number of columns in argument arrays

        searchCols        - names of the columns to compare

        minValues         - min/max values for comparison
        maxValues

        maxRows           - max number of rows to find


search(const char* filename, const char* searchCol, const char* value,
       int maxRows);

        Search the given tab table file for upto maxRows rows where
        the given column has the given value and fill this table with
        the resulting rows.

        Args:

        filename         - tab table file to search

        searchCol        - name of the column to compare

        value            - value for comparison

        maxRows          - max number of rows to find


search(const char* filename, int searchCol, const char* value,
       int maxRows);

        Search the given tab table file for upto maxRows rows where
        the given column has the given value and fill this table with
        the resulting rows.

        Args:

        filename         - tab table file to search

        searchCol        - index of the column to compare (0..n)

        value            - value for comparison

        maxRows          - max number of rows to find
```

printRows(ostream& os)
Print the table rows to the given stream in tab table format.

friend ostream& operator<<(ostream& os, TabTable& t)
Output operator, prints the table to the given stream in tab table format.

numRows() const
Return the number of rows in the table.

void numRows(int n)
Set the number of rows in the table.

numCols() const
Return the number of columns in the table.

colNames() const
Return an array of column names in the table.

status() const
Return the status after the constructor has completed.

SEE ALSO

QueryResult(3)

- - - - - -
Last change: 02 Feb 98

4.0.20 TcsCatalog(3)

NAME

TcsCatalog - class for accessing TCS catalogs

SYNOPSIS

```
#include "TcsCatalog.h"

class TcsCatalog : public AstroCatalog {
public:
    TcsCatalog(const CatalogInfoEntry* e);
    virtual ~TcsCatalog();

    static TcsCatalog* open(const char* name);

    int numCols();
    char** colNames();
    const char* colName(int col);
    int colIndex(const char* colName);
    int hasCol(const char* name);

    int getObject(const char* id, TcsCatalogObject& obj);

    int searchClosestStar(const WorldCoords& pos,
        double mag0, double mag1,
        TcsCatalogObject& obj);
};
```

DESCRIPTION

This class is a subclass of AstroCatalog(3) specialized for accessing GSC, PPM or similar catalogs for use by the TCS (Telescope Control Software). A TcsCatalog is like an AstroCatalog, except that it assumes a catalog has fixed columns, such as those that are found in the GSC or PPM catalogs. This class restricts itself to these fixed columns and ignores the rest. Each row of a TcsCatalog can be represented by a TcsCatalogObject. Any missing column values are set to the appropriate null value (defined in TcsCatalogObject.h).

CATALOG SERVERS

This class uses the same catalog servers as the AstroCatalog class. Please refer to AstroCatalog(3) for details.

LOCAL CATALOGS

The format of local catalog files and query results that are stored in files is the same here as in AstroCatalog(3), except that here the column names are fixed. In addition, catalogs saved in TCS format contain additional comments in the header indicating that the catalog is a TCS catalog and with the meanings of the columns.

UNITS

Unless otherwise stated, the units for all radius values are in arcmin. Floating point values for RA and DEC are always in degrees. The default equinox is J2000.

QUERY PARAMETERS

Parameters for catalog searches are checked to make sure they are valid and in the correct range. A radius value must be between 0.0 and 300.0 arcmin. A magnitude may have any value. For ranges, such as min and max radius and min and max magnitude, the order of the arguments is not important, since they will be rearranged as needed. If both min and max values are 0.0, they are ignored for that search. If you really want to search for objects with "0.0" magnitude, specify a range with small values that are not zero, for example, min/max mag = (-0.0001, +0.0001).

PUBLIC METHODS

```
TcsCatalog(const CatalogInfoEntry* e)
        Constructor - create a TCS catalog class instance.  Note: the
        public interface normally uses TcsCatalog::open().  The
        argument represents the entry in the catalog config file for
        this catalog.

~TcsCatalog()
        Destructor - close catalog and free any resources.

open(const char* name)
        Open the named catalog and return a pointer to a new
        TcsCatalog (or derived class) object created for it or NULL if
        errors occur. If the given name is really the name of a file,
        it is opened as a local catalog and the pointer returned is of
        type TcsLocalCatalog, a subclass of TcsCatalog. If the file
        is not in the correct format, a NULL pointer is returned.

numCols()
        Return the number of columns in the catalog (fixed number).

colNames()
        Return an array of column names (this is always the same for
        TCS catalogs).

colName(int col)
        Return the name of the given column (fixed for TCS catalogs).

colIndex(const char* colName)
        Return the column index for the given column name (fixed for
        TCS catalogs).

hasCol(const char* name)
        Return true if the catalog contains the given column (fixed
        for TCS catalogs).

int getObject(const char* id, TcsCatalogObject& obj)
        This method is redefined in this class to get the object given
        by "id" in the catalog and return 0 if all is OK.  Note that
        the interface is changed here to use the TcsCatalogObject to
        hold the result rather than a QueryResult as in the parent
        class version.

        Arguments:

        id,             (in ) - object id in catalog.
        obj             (out) - object for row, if found.

int searchClosestStar(const WorldCoords& pos, double mag0, double mag1,
                      TcsCatalogObject& obj)
        This method is redefined in this class to search for the star
        closest to the given position, with the magnitude in the given
        range. Note that the interface is changed here to use the
```

```
TcsCatalogObject to hold the result rather than a QueryResult
as in the parent class version.

Arguments:

pos,            (in )  - center position in world coordinates
mag0,           (in )  - min magnitude
mag1,           (in )  - max magnitude
obj             (out)  - object for row, if found
```

SEE ALSO

`AstroCatalog(3), TcsCatalogObject(3), TcsQueryResult(3), TcsLocalCatalog(3)`

```
- - - - - -
Last change: 02 Feb 98
```

4.0.21 TcsCatalogObject(3)

NAME

```
TcsCatalogObject - Class representing one row of results of a
                   TcsCatalog query.
```

SYNOPSIS

```
#include "TcsCatalogObject.h"

// NULL values
#define TCS_CATALOG_NULL_INT 4294967294      /* (2^32 - 1) */
#define TCS_CATALOG_NULL_DOUBLE 1.E-300

class TcsCatalogObject {
...
public:
    TcsCatalogObject();
    TcsCatalogObject(const TcsCatalogObject&);
    ~TcsCatalogObject();

    TcsCatalogObject& operator=(const TcsCatalogObject &);
    friend ostream& operator<<(ostream&, const TcsCatalogObject&);

    void print(char* buf, int bufsize);
    static void printHeadings(ostream& os);
    static void printHeadings(char* buf, int bufsize);
    int printTableRow(ostream&);

    void reset();

    static int isNull(int v);
    static int isNull(double v);
    static int isNull(const char* v);

    // set values
    int id(const char*);
    int ra(double);
    int dec(double);
    int cooSystem(const char*);
    int epoch(double);
    int pma(double);
    int pmd(double);
    int radvel(double);
    int parallax(double);
    int cooType(const char*);
    int band(const char*);
    int mag(double);
    int more(const char*);
    int preview(const char*);
    int distance(double);
    int pa(double);

    // get values
    const char* id();
    double ra();
    double dec();
    const char* cooSystem();
    double epoch();
    double pma();
    double pmd();
    double radvel();
    double parallax();
```

```
    const char* cooType();
    const char* band();
    double mag();
    const char* more();
    const char* preview();
    double distance();
    double pa();

    static int numCols();
    static char** colNames();
    static const char* colName(int col);
    static int colIndex(const char* colName);
    static int hasCol(const char* name);
};
```

DESCRIPTION

```
This object represents the contents of one row of TCS (GSC or PPM)
query results.  Missing values are set to the appropriate null value
(see above). Methods are available to set and get the object values.
The "set" methods do some error checking. The "get" methods are all
simple "inline" methods that simply return the value.

The following column values represent a TCS catalog object:

 id             object catalog id
 ra             Alpha coordinate for the target in decimal degrees
 dec            Delta coordinate for the target in decimal degrees
 cooSystem      Equinox system and equinox ("B1950" or "J2000")
 epoch          Epoch expressed as decimal year
 pma            Proper Motion Alpha in radians/year (-10.0 to 10.0)
 pmd            Proper Motion Delta in radians/year (-10.0 to 10.0)
 radvel         radial velocity in km/sec (-200000 to 200000)
 parallax       Parallax in arcseconds (-10000 to 10000)
 cooType        Coord. type: "M" for mean, "A" for apparent character
 band           Magnitude wavelength band ("V")
 mag            Object's magnitude in given band
 more           An HTTP URL pointing to more info on the object
 preview        An HTTP URL pointing to an image of the object
 distance       distance to center of the field
 pa             position angle based on center of the field
```

UNITS

```
Unless otherwise stated, the units for all radius values are in arcmin.
Floating point values for RA and DEC are always in degrees. The
default equinox is J2000.
```

PUBLIC METHODS

```
TcsCatalogObject()
        Constructor: initialize all fields to null.

TcsCatalogObject(const TcsCatalogObject&)
        copy constructor.

~TcsCatalogObject()
        Destructor.

TcsCatalogObject& operator=(const TcsCatalogObject &)
        Assignment.

operator<<(ostream&, const TcsCatalogObject&)
```

```
        Output operator (Tcl list format).

print(char* buf, int bufsize)
        Print this object to the given buffer.

printHeadings(ostream& os)
printHeadings(char* buf, int bufsize)
        Print the headings to match the output of '<<' above.

printTableRow(ostream&)
        Print this object as a tab separated row

reset()
        Result all fields to default values.

isNull(int v)
isNull(double v)
isNull(const char* v)
        Return true if the given value is null (by convention).

id(const char*)
ra(double)
dec(double)
cooSystem(const char*)
epoch(double)
pma(double)
pmd(double)
radvel(double)
parallax(double)
cooType(const char*)
band(const char*)
mag(double)
more(const char*)
preview(const char*)
distance(double)
pa(double)
        Set fields, with range checking, return 0 if OK.

id()
ra()
dec()
cooSystem()
epoch()
pma()
pmd()
radvel()
parallax()
cooType()
band()
mag()
more()
preview()
distance()
pa()
        Member access: return member value.

The following methods are for compatibility with other AstroCatalog
classes:

numCols()
        Return the number of columns in the catalog.

colNames()
```

```
        Return an array of column names.

colName(int col)
        Return the column name for the given column.

colIndex(const char* colName)
        Return the column index for the given column name.

hasCol(const char* name)
        Return true if the catalog contains the given column.
```

SEE ALSO

TcsCatalog(3), TcsQueryResult(3)

\- - - - - -

Last change: 02 Feb 98

4.0.22 TcsQueryResult(3)

NAME

TcsQueryResult - class for accessing results of a TCS catalog query

SYNOPSIS

```
#include "TcsQueryResult.h"

class TcsQueryResult : public QueryResult {
...
public:
    TcsQueryResult();
    TcsQueryResult(const char* result);
    virtual ~TcsQueryResult();

    int getObj(int row, TcsCatalogObject&);

    int printRows(ostream& os);

    // -- redefine these to deal with TCS columns --

    char** colNames();
    int numCols();
    const char* colName(int col);
    int colIndex(const char* colName);
};
```

DESCRIPTION

This class manages the result of an TcsCatalog::query. Internally a QueryResult object manages the "tab table" query results. This class provides transparent access to the result based on a row, column index and, unlike the QueryResult class, is specialized for accessing the TCS (GSC and PPM) catalogs. There are special methods to return GSC and PPM fields in common units and to determine if a field is present in the catalog.

UNITS

Unless otherwise stated, the units for all radius values are in arcmin. Floating point values for RA and DEC are always in degrees. The default equinox is J2000.

PUBLIC METHODS

TcsQueryResult()
    Constructor: initialize empty table.

TcsQueryResult(const char* result)
    Constructor: initialize from query result buffer.

~TcsQueryResult()
    destructor: free any allocated memory.

getObj(int row, TcsCatalogObject&);
    Access a TcsCatalog (GSC/PPM) result row: fill out the given TcsCatalogObject.

printRows(ostream& os);
    Print table rows to the given stream.

```
colNames()
        Get array of column names in a TCS query result.

numCols()
        Get number of columns in a TCS query result.

colName(int col)
        Return the column name for the given column index.

int colIndex(const char* colName)
        Return the column index for the given column name
```

SEE ALSO

TcsCatalog(3), TcsCatalogObject(3)

\- - - - - -

Last change: 02 Feb 98

4.0.23

5 INSTALLATION

5.1 Before you build the CAT software

The cat package depends on the following packages being already installed. These are available from ftp://ftp.archive.eso.org/pub/skycat:

- tclutil - Tcl and C++ Utilities Package
- astrotcl - Astronomical Tcl and C++ Utilities Package

These packages also assume the following standard Tcl packages are installed:

- itcl-2.2 - [Incr Tcl], includes tcl7.6, tk4.2
- BLT-2.1 - BLT toolkit, graphs and other widgets
- tclX-7.6.0 - Extended Tcl

These packages are available from the TCL archives.

See: http://www.tcltk.com/ for Itcl.

and http://www.NeoSoft.com/tcl/ for TclX and other contributed Tcl software

You can also find a tar file on ftp://ftp.archive.eso.org/pub/skycat that contains all of the Tcl packages used, with any required patches installed.

In addition, the catalog library requires the *Tix* package, which is used for the catalog directory browser:

- Tix4.1.0 - Tix widget set

Tix is also avilable from the above URLs.

You can install the standard Tcl/Tk packages anywhere, however it is easiest to install them all in the same directory (using the same -prefix argument to configure) and define the `TCLTK_ROOT` environment variable to point to the top level install directory. This variable is used, if defined, by the *tclutil*, *astrotcl* and *cat* package configure scripts, to locate the Tcl installation.

If you are building the *tclutil*, *astrotcl* or other related packages also, it is best to place these packages under the same parent directory, so that you have $dir/tclutil, $dir/astrotcl, and $dir/cat (so there is no problem finding the local include files and libraries).

5.2 Build the CATLIB Software

To build the software, type:

```
configure
make
make install
```

The default install dir is /usr/local. You can specify the -prefix argument to configure to change this:

```
configure -prefix $INSTALLDIR
```

The compilers used are the same as those used to build the *tclutil* package. The file `tclutilConfig.sh` is read by configure to get the necessary information.

If you specify the configure option --enable-shared:

```
configure -prefix $INSTALLDIR --enable-shared
```

you can build a shared library and load the Tcl commands with the Tcl *load* command or load it as a Tcl package under the name *Cat*. The Tcl code to do this would look something like this, assuming the pkgIndex.tcl file for the *Cat* package is in your Tcl *auto_path* variable:

```
if {[catch {package require Cat} msg]} {
 puts "error loading Cat package $msg"
 exit 1
}
```

5.3 VLT Make Procedure

As an alternative to running configure and make, you can also do this:

```
cd src
make
make install
```

The Makefile in the cat/src directory runs configure and then make as described above. You can also specify options to that Makefile, for example:

```
cd src
make PREFIX=$INSTALLDIR CONFIGURE_FLAGS=--with-gcc
```

The PREFIX variable defaults to /usr/local and is the prefix of the directory in which to install the software.

5.4 Running the tests

To test the software you can run the tests in any of the "test" directories. For example, to test the C++ classes:

```
cd astrocat/test
make test
```

For the C wrappers:

```
cd ccat/test
make test
```

For the Tcl scripts:

```
cd tclcat/demos
astrocat
```