SSN/75.1

Starlink Project
Starlink System Note 75.1

A.C. Davenhall

26 July 2000

Writing Catalogue and Image Servers for GAIA and CURSA

Abstract

GAIA and CURSA can interrogate remote catalogues via the Internet to return lists of objects which satisfy a given criterion. GAIA can also extract images of a specified region of sky from remote databases. Similar facilities are also available in other packages, such as the ESO *SkyCat* image display tool and the Gemini observing tool. This functionality is achieved by having servers running on the remote systems which accept queries sent by GAIA *etc*, interrogate their local copies of the catalogues to find the data which satisfy the query and return them. The servers can communicate with GAIA *etc.* because the query is sent, and the results returned, in a standard format. This document describes how to write such servers.

Who Should Read this Document?

This document is aimed at programmers who intend to write servers which will provide catalogue and image data for GAIA *etc*.

Contents

Part I

Preface

Accessing this document

A hypertext version of this document is available. To access it on Starlink systems type:

```
showme   ssn75
```

On non-Starlink systems access URL:

```
http://www.starlink.ac.uk/docs/ssn75.htx/ssn75.html
```

Paper copies can be obtained from the Starlink document librarian, who can be contacted as follows.

Postal address:
The Document Librarian. Starlink Project, Rutherford Appleton Laboratory, Chilton,
DIDCOT, Oxfordshire, OX11 0QX, United Kingdom.

Electronic mail: `starlink@jiscmail.ac.uk`

Fax:

from within the United Kingdom:	01235-445-848
from overseas:	+44-1235-445-848

Obtaining assistance

If you run into difficulties writing a server then I might be able to offer advice or assistance. I can be contacted as follows.

Postal address:
A.C. Davenhall. Institute for Astronomy, Royal Observatory, Blackford Hill,
Edinburgh, EH9 3HJ, United Kingdom.

Electronic mail: `acd@roe.ac.uk`

Fax:

from within the United Kingdom:	0131-668-8416
from overseas:	+44-131-668-8416

Revision history

(1) 26th July 2000: Version 1. Original version (ACD).

List of Figures

List of Tables

1 Introduction

GAIA[4] and CURSA[3] can interrogate remote catalogues via the Internet to retrieve lists of objects which satisfy a given criterion. GAIA can also extract images of a specified region of sky from remote databases. Similar facilities are also available in other packages, such as the ESO *SkyCat*[1] image display tool and the Gemini observing tool. This functionality is achieved by having servers on the remote systems which accept queries sent by GAIA *etc*, interrogate their local copies of the catalogues to find the data which satisfy the query and return them to the client which sent the query. This method of working is an example of a 'client-server architecture', with GAIA, CURSA *etc.* acting as the client. The server and client can communicate because the query is sent, and the results returned, in a standard format. This document describes how to write such servers and also the formats of the queries and results.

The standard formats for the queries and returned data were developed by Allan Brighton and colleagues at ESO for use with their Astronomical Catalogue Library (ACL). This subroutine library provides the functionality for a client to access a remote server, and is used by, for example, *SkyCat* and GAIA. Consequently, in this document the formats will be called the 'ACL format' and a server which accepts queries and returns data in these formats will be called an 'ACL server'.

The communication between the client and the server uses the Hyper-Text Transfer Protocol (HTTP) developed as part of the World Wide Web. The servers are, strictly speaking, *gateways* using the Common Gateway Interface (CGI). One way of thinking of the client is as a very specialised Web browser. A consequence of this approach is that if a site is to host an ACL server it must also be running a Web server. The HTTP and CGI protocols are, of course, enormously flexible and the ACL format is a set of additional rules and restrictions which sit 'on top' of them.

An ACL server is a CGI gateway, and CGI gateways can be provided in any number of different ways. However, the usual technique is to implement the gateway as a Perl script and this approach will be adopted in this document. Usually (though not always) the Perl script will invoke a special-purpose program or Database Management System (DBMS) to interrogate the catalogue database.

This document is divided into two parts:

Part I – a tutorial example of creating a simple ACL server,

Part II – reference material, mostly describing the ACL format.

2 Further Reading

The ACL format is documented in Section 2 of Allan Brighton's *Astronomical Catalogue Library User Manual*[2] (and the description in Part II is largely based on this manual). It is a subset of a proposed general format for exchanging information between remote astronomical information services which is being developed at the Centre de Données astronomiques de Strasbourg (CDS) and elsewhere. The full proposal is described in the working document *Astronomical Server URL* by M. Albrecht *et al.*[1].

There are numerous books describing the HTTP and CGI protocols; I have found the *The HTML Source Book* by Ian Graham[5] to be useful. Similarly, there are many books on Perl. I have used *Learning Perl* by Randall Schwartz[7] and *Programming Perl* by Larry Wall and Randall Schwartz[8] and found them comprehensive, convenient and accessible.

[1]http://archive.eso.org/skycat/

Part II

Tutorial Example: Creating a Simple Server

3 Introduction

This part of the document is a tutorial example which describes how to create a simple ACL server. Example files are provided which illustrate the procedure. Firstly, however, the basics of using an ACL server to query a catalogue are reviewed.

4 Basics of Querying Remote Catalogues

The basics of querying a catalogue are that some criterion is specified, all the rows in the catalogue are examined and those which satisfy the criterion are returned as the list of selected rows. Some criteria might be:

- stars whose Right Ascension lies in the range $13^{h}\,30^{m}.0$ to $14^{h}\,00^{m}.0$,
- stars brighter than $m_v = 13.0$,
- stars with a B-V colour greater than 0.2.

However, a very common sort of search on astronomical catalogues is the so-called 'circular area search' or 'cone search'. Virtually all astronomical catalogues contain celestial coordinates, in practice Right Ascension and Declination for some equinox and epoch. In a circular area search the central coordinates and angular radius are specified. All the objects in the catalogue which are less than this angular radius from the central coordinates are selected. That is, the circular area search finds all the objects in the catalogue within a given circular patch of sky.

It is usual for ACL servers to a support a circular area search and often this may be the only type of search provided. The full form of an ACL circular area search is slightly more general, with both an inner and outer radius specified, so that objects inside an annulus rather than a circle are selected. (The 'traditional' circular area search corresponds to setting the inner radius to zero.)

The catalogue being accessed will doubtless have other columns as well as the Right Ascension and Declination, and the ACL server may permit 'range' searches on some of these columns. In a range search minimum and maximum values are specified for a column and a row is selected if its value for the column falls within the given range. Any range searches specified are combined with each other and with the circular area search using a 'logical and'. That is, for example, the objects selected would correspond to those which are both in a given area of sky and in a given magnitude range. Though this mechanism allows powerful queries to be made it still provides only a subset of all the conceivable types of queries.

Before starting work on an ACL server for a catalogue, you need to decide two things.

(1) Are circular area searches to be supported (the answer is almost certainly yes)?

(2) On which, if any, additional columns are range searches to be supported?

In order to implement the server you need to provide two things:

(1) the server itself,

(2) an entry for the server in an ACL 'configuration file'.

An ACL configuration file defines the list of catalogues which a client, such as GAIA or *SkyCat*, currently knows about. It has an entry for each catalogue. The entry specifies details of the catalogue which the client needs to know: the URL of its server, the types of queries that it supports, the name by which it is to be described to the user *etc*.

5 Obtaining Example Files

Some simple examples of server scripts and a configuration file are provided with this document. Subsequent sections of the tutorial will describe these files and explain how to install them. You may also find them useful as templates for developing your own servers and configuration files. The files can be obtained in two different ways.

- On Starlink systems they should be present in directory:

  `/star/examples/ssn75`

- Copies can be obtained by anonymous ftp from Edinburgh. The details are:

Anonymous ftp to:	`ftp.roe.ac.uk`
Directory:	`/pub/acd/misc`
File:	`ssn75_examples.tar`

  The file is an uncompressed tar archive (it is only 52 Kb in size). Set ftp to 'binary' mode to retrieve it.

The important files used in the tutorial are:

`simpleserver.cgi` a simple server script,

`secondserver.cgi` a second, slightly more complicated, server script,

`genfield.c` a program to generate the list of stars returned by the servers,

`simpleconfig.cfg` a configuration file including the two example servers.

`checkcfg` a script for checking configuration files.

File `0README.LIS` gives a complete list. You will probably find it useful to print out copies of the files and have them to hand as you work through the examples. Copies of the servers are installed at Edinburgh, so the URLs given in the examples should work, though you will be accessing the Edinburgh versions rather than your own copies.

6 Creating a Server

An ACL server is a Perl script which accepts a query in a standard format, searches the corresponding catalogue to find the objects which satisfy the query and returns them to the remote client. The server `simpleserver.cgi` supplied with this document is more-or-less the simplest practical server, but it illustrates the structure that they usually have. You will find it helpful to have a copy to hand as you read through this section. `simpleserver.cgi` supports only circular area searches in which a central Right Ascension and Declination and an angular radius are specified. Because it is provided as an example it generates and returns an artificial list of stars, centred on, and scaled to fit, the specified area, rather than searching a real catalogue. The overall structure of `simpleserver.cgi` is:

```
obtain the query string
parse the query string to obtain the central coordinates and radius
generate the list of stars to fit in this field
    (a real server would search a catalogue instead)
return the results to the client
```

The comments in the source code should make the detailed working obvious. However, the following notes about the query string and results returned might be useful.

6.1 Query string

(1) If a CGI gateway is implemented as a Perl script the query string appears in the variable `$ENV{'QUERY_STRING'}`. In `simpleserver.cgi` the query is copied into a local variable by the line:

```
$query = $ENV{'QUERY_STRING'};
```

(2) In `simpleserver.cgi` the query string has the format:

`ra=`*xxx*`&dec=`*yyy*`&radius=`*zzz*

For example:

```
ra=10:30:00&dec=-30:30.0&radius=3
```

where `ra=`, `dec=` and `radius=` have their obvious meanings. When you set up the entry for the server in the configuration file (see Section 7, below) you have considerable latitude over this format. However, the one used by `simpleserver.cgi` is a common one, and is as good as any. `simpleserver.cgi` supports only circular area searches; if additional types of search were supported then the query string would contain additional parameters.

(3) The formats and units required for the parameters are as follows:

Right Ascension sexagesimal or decimal hours,

Declination sexagesimal or decimal degrees,

Radius decimal minutes of arc.

If a sexagesimal value is entered then a colon (':') should be used as a separator. The Right Ascension and Declination should be for equinox and epoch J2000. The server *must* accept values in these units and formats because typically the client will present the same recipe and information to the user when requesting input for any of the various catalogues available to it.

6.2 Results returned

(1) The first line of information returned by the server should always be the MIME type. In `simpleserver.cgi` it is written by the line:

```
{  print "Content-type: text/plain\n\n\n";
```

The MIME type is not part of the table of results, but rather is used by the client or Web browser to interpret the format of the data which follows.

(2) The list of selected objects are returned to the client as a stream of ASCII characters. The list is written in the Tab-Separated Table (TST) format (see Section 11).

(3) The Perl script should simply write the results to standard output (whence it will be automatically forwarded to the remote client). In `simpleserver.cgi` the lines:

```
$tst = `$queryExe $ra $dec $radius`;
print "$tst";
```

invoke program `genfield` (variable `$queryExe` has previously been set to contain the name and directory specification of the executable for `genfield`) to generate the star list, copy the list the to variable `$tst` and then write the contents of `$tst` to standard output. (Hint: Perl has several mechanisms for invoking processes and directing their output to standard output; I found the one described to be the most suitable for use in a CGI script.)

(4) The last line written to standard output should be:

```
print "[EOD]\n";
```

The string '`[EOD]`' informs the client that the server has finished sending data.

6.3 Installing and testing the server

The procedures for installing CGI scripts vary at different sites; your system manager should be able to advise on arrangements at your site. However, to install `simpleserver.cgi` you need to follow at least the following steps.

(1) Copy files `simpleserver.cgi` and `genfield.c` to a suitable directory (there are likely to be restrictions on which directories can contain CGI scripts; see your system manager).

(2) Compile program `genfield.c` and name the executable `genfield`. `genfield.c` is a standard (and simple) C program; any C compiler should be able to handle it.

(3) Edit file `simpleserver.cgi`. Locate the line:

```
$queryExe = "/star/examples/ssn75/genfield";
```

which is towards the top of the script and change it to correspond to whichever directory you have put the files in.

(4) Check with your system manager whether there are any other local requirements for running CGI scripts.

(5) You are now ready to test the server. The tests are best conducted from a normal Web browser, such as `netscape`. Start the browser and enter a URL similar to:

```
http://www.roe.ac.uk/~acd/cgi-bin/simpleserver.cgi?ra=10.0&dec=30.0&radius=3
```

This string comprises the normal URL for the CGI script, followed by a '?', followed by the query. This format is the normal syntax for invoking CGI scripts and passing queries to them. To invoke your version of the server you would substitute the appropriate URL in the example above. Typing in the example exactly as given should invoke a version of the server running in Edinburgh. If all is working correctly the browser should display a table similar to the one in Figure 1. If the server fails then your Web server probably maintains error logs which might contain useful diagnostics; your system manager should know where these logs are kept.

```
Example Star Field.

#column-units:          DEGREES         DEGREES         Magnitudes
#column-types:   CHAR*8         DOUBLE  DOUBLE  REAL
#column-formats: A8     F12.6   F12.6   F6.2

Id      RA      DEC     mag
--      --      ---     ---
Star 1  150.026667      30.050000       0.580000
Star 2  149.968333      29.961667       0.120000
Star 3  149.983333      30.041667       1.640000
Star 4  149.993333      30.005000       2.210000
Star 5  150.000000      30.000000       1.700000
Star 6  150.006667      29.996667       1.790000
Star 7  149.983333      29.993333       3.380000
Star 8  150.015000      29.951667       4.130000
[EOD]
```

Figure 1: The table of values returned by the ACL server `simpleserver.cgi`.

6.4 Modifying the server

Other servers are likely to be similar to `simpleserver.cgi`. To modify it to access your own catalogue you would replace the invocation of program `getfield` with an invocation of a program which searched your catalogue or DBMS.

`simpleserver.cgi` is more-or-less the simplest practical server, and is provided as an example. Additional useful features in a server include: copying the queries to a log file (so that you can monitor usage) and copying error messages to a second log file (as diagnostics in case of misadventure). Another server, `secondserver.cgi`, which incorporates these features, is supplied with this document. To install it, follow the same procedure as for `simpleserver.cgi`, except that the lines towards the top of the script which need to be modified are:

```
$queryExe = "/star/examples/ssn75/genfield";
$logDir = "/star/examples/ssn75/examplelogs";
```

When developing a server it is often useful to comment out the line:

```
        $query = $ENV{'QUERY_STRING'};
```

and un-comment the line:

```
#    $query = "ra=10:30:00&dec=-30:30.0&radius=3";
```

so that the script has a query 'hard-wired'. It can now be run directly from the command line rather than be invoked via a Web browser. This trick often makes debugging scripts a *lot* easier.

Both the query and error log files written by `secondserver.cgi` are simple text files. Note, however, that file `query.TXT`, which is supplied with the examples, allows the query log to be accessed by CURSA[3] as an STL (Small Text List) format catalogue. Once the query log becomes large you might find it more convenient to examine it with the CURSA catalogue browser `xcatview` rather than Unix commands such as `cat` and `more`.

7 Creating a Configuration File

The configuration file used by GAIA *etc.* mediates interaction between the client and server. It is an ASCII text file containing details for each of a list of catalogues. GAIA (or whatever) reads the file and the catalogues listed in it become the ones that GAIA knows about. The details supplied for each catalogue are things like: its URL, the type of queries supported, the name that will be used to describe it to users *etc*. The entry for a typical simple catalogue looks something like:

```
serv_type:      catalog
long_name:      Simple example server.
short_name:     simple@roe
url:            http://www.roe.ac.uk/~acd/cgi-bin/simpleserver.cgi?ra=...
symbol:         mag circle 3
```

This entry is taken from `simpleconfig.cfg`, the example configuration file supplied with this document, though the `url` entry has been truncated. By convention configuration files have file type '`.cfg`'. The purposes of the various items are as follows.

`serv_type:` is the type of the server. For a straightforward catalogue the value required is '`catalog`'. Other values are possible, though you will probably rarely use them.

`long_name:` a one-line name or short description of the catalogue. It will be presented to the user to allow him to identify the catalogue.

`short_name:` a short name for the catalogue. Conventionally it has the form:

catalogue@*institution*

where *catalogue* is an abbreviation for the catalogue and *institution* a standardised abbreviation for the institution where the on-line version is located. By convention *institution* has three or four characters.

`url:` the URL used to access the server. Following the usual conventions for a CGI gateway it consists of the URL corresponding to the script which constitutes the server, followed by a '`?`' and then a string defining the query passed to the server (see Section 7.1).

`symbol:` specifies how objects are to be plotted (see Section 10.6).

There are various other optional items which can be included. They are described in Section 10.

7.1 URL query

The string appended to the server URL in the configuration file and which defines the type of queries supported by the catalogue has a format something like:

```
ra=%ra&dec=%dec&radius=%r2
```

It consists of simple characters and 'tokens'. The tokens start with a '%' character. When GAIA makes a query the tokens are replaced with values which correspond to the individual query and the resulting string is sent to the server. For example, tokens in the above string might be susbstituted to yield:

```
ra=10:30:00&dec=-30:30.0&radius=3
```

Obviously the format of the query string appended to the URL in the configuration file must correspond to that expected by the server. Various standard tokens can be included in the query string. Some common ones are:

`%ra`	Right Ascension,
`%dec`	Declination,
`%r1`	inner radius,
`%r2`	outer radius,
`%n`	maximum number of objects to return.

For a complete list see Section 10.

7.2 Installing and testing the configuration file

Installing and testing the configuration file should be quite straightforward. The servers `simpleserver.cgi` and `secondserver.cgi` should be installed (see Section 6, above). Then proceed as follows.

(1) Edit the configuration file and change the URLs for the servers `simple@roe` and `second@roe` to correspond to wherever you have installed the servers.

(2) A Perl script to check a configuration file for errors is included with the example files for this document. To check that you have not introduced any errors whilst editing the example configuration file type:

```
/star/examples/ssn75/checkcfg   simpleconfig.cfg
```

(If the example files are not in their standard location on a Starlink system then obviously you need to alter the directory specification accordingly. Also, on non-Starlink systems you might need to edit the first line of `checkcfg` to correspond to wherever Perl is installed on your system.) If the configuration file is valid then `checkcfg` will report:

```
Configuration file parsed successfully.
```

Conversely, if it contains errors then messages describing the problems will be reported. `checkcfg` is described in Section 10.7.

(3) Start GAIA and test the server. To import the configuration file into GAIA click on the Data-Servers button on the right hand side of the main menu bar and choose the Browse Catalog Directories option. A window showing the catalogues available should appear. Click on the File button at the left of its top menu bar and choose the Load Config file... option. A window allowing you to select the required file should then appear.

Once the configuration file has been loaded you can choose from amongst its catalogues and make selections in the normal fashion. If you make a selection from the simple example server a list of objects similar to Figure 1 should be returned.

7.3 Modifying the configuration file

When you create a new server you need to create an entry for it in your configuration file. If the server is just a simple variation of `simpleserver.cgi` or `secondserver.cgi` and only supports circular area queries then just duplicate the entry for `simple@roe` and change `long_name`, `short_name` and `url` to correspond to your server.

Additional modifications can be made as required. The following section gives some examples and the options are documented in Section 10. Remember that script `checkcfg` (see Section 10.7) is available for checking configuration files.

8 Carrying On

This section introduces a few additional features that are often required in servers and configuration files.

8.1 Multiple catalogue servers

`simpleserver.cgi` can access only a single catalogue. Often you may want to write a server which can access each of several catalogues. In this case your configuration file must contain an entry for each *catalogue*, not each server. An additional parameter, whose value identifies the catalogue, is added to the query. The server parses this parameter to identify the catalogue required. You can decide the syntax and values of this parameter, though the configuration file and server must agree.

For example, suppose that you are writing a server which will provide access to the SAO, PPM and Hipparcos astrometric catalogues. You might invent a parameter called '`catalogue`' whose value identifies the catalogue. Your configuration file would then have three entries like:

```
serv_type:  catalog
long_name:  SAO (Smithsonian Astrophysical Observatory) catalog
short_name: sao@roe
url:        http://www.roe.ac.uk/~acd/cgi-bin/ast.cgi?catalogue=sao&ra=%ra&dec=...
symbol:     mag square 3

serv_type:  catalog
long_name:  PPM (Positions and Proper Motions) catalogue
short_name: ppm@roe
url:        http://www.roe.ac.uk/~acd/cgi-bin/ast.cgi?catalogue=ppm&ra=%ra&dec=...
symbol:     mag square 3

serv_type:  catalog
long_name:  Hipparcos catalogue
short_name: sao@roe
url:        http://www.roe.ac.uk/~acd/cgi-bin/ast.cgi?catalogue=hipparcos&ra=%ra&dec=...
symbol:     mag square 3
```

The server would be written to parse the value of `catalogue` and then search the catalogue indicated.

8.2 Providing range queries

As well as selecting objects in a circular area of sky it is possible to further restrict the objects selected to only those for which the values of some column lie within a given range. Suppose selections from an optical catalogue were to be optionally restricted to also lie within a specified range of magnitudes and the column of magnitudes was named `mag`. To provide this facility for a catalogue its entry in the configuration file should be include the keyword `search_cols` and the query part of its `url` keyword should include the token `%cond`.

(1) The syntax of the `search_cols` keyword is:

`search_cols:` *column-name minimum-prompt maximum-prompt*

For example:

```
search_cols:  mag "Bright limit" "Faint limit"
```

column-name is the name of the column. The client uses *minimum-prompt* and *maximum-prompt* as prompts when soliciting the extrema of the required range from the user. (Note that because the example column is a magnitude the "Bright limit" corresponds to the smallest numerical value and the "Faint limit" to the largest.)

(2) The `%cond` token is added to the query part of the `url` keyword to supply details of the range query. Such a query string might look like:

```
ra=%ra&dec=%dec&radius=%r2&%cond
```

The syntax of the values substituted into the `%cond` string is:

column-name = *minimum-value* , *maximum-value*

For example, if a range of first to second magnitude had been specified for column `mag` then `%cond` would translate to:

```
mag=1.0,2.0
```

The server that parses the query must interpret this string and ensure that the range selection specified is applied.

See Section 10.5 for further details.

8.3 Linked configuration files

In addition to entries for individual catalogues, configuration files can also contain entries for other configuration files. Typically when such an entry is chosen all the entries in the target configuration file are loaded into the client. In this way a tree (or rather a network, because recursion is allowed) of entries can be built up. Entries of this type are referred to as 'directories' (by analogy with an hierachical file system).

For a directory entry the `serv_type` should be '`directory`' and the `url` should be the URL of the destination configuration file. `long_name` and `short_name` have their usual meaning. Other options are unlikely to be required. An example might be:

```
serv_type:      directory
long_name:      ESO Catalogues
short_name:     catalogs@eso
url:            http://archive.eso.org/skycat/skycat2.0.cfg
```

8.4 Handling queries which return no results

Sometimes users will submit a query which no objects in the catalogue satisfy. For example, the query might correspond to an empty patch of sky. In practice such queries are quite common. Unfortunately, the ACL format does not prescribe the action the server should take in this case. However, the *recommended* action is for the server to return an empty TST table. That is, it should return all the header information for a TST generated from the catalogue being queried (see Section 11), down to and including the list of column names and the line of dashes anf tab characters which terminate the header, but no table of values.

Part III

Reference Material

9 Introduction

This part of the document comprises reference material describing the ACL configuration file and the Tab-Separated Table (TST) format. Most of the material is adapted from Section 2 of Allan Brighton's *Astronomical Catalogue Library User Manual*[2].

10 The Configuration File

This section describes the format of the ACL configuration file. When you refer to it you might find it useful to have to hand a copy of either `simpleconfig.cfg` or some other configuration file.

An ACL configuration file mediates the interaction between a client such as GAIA and a remote server. The configuration file comprises a list of one or more databases, giving details for each. Usually each 'database' will be a simple astronomical catalogue. However, other alternatives are possible: archives, name servers, *etc*. Consequently, in this section the generic term 'database' is used to denote each entry. Also, it is individual databases, not servers, which are listed in the configuration file: some servers might give access to more than one database. The details supplied for each database are things like: its URL, the type of queries supported, the name that will be used to describe it to users *etc*. The client reads the configuration file and the databases listed become the ones that it knows about.

By convention, configuration files have file-type '`.cfg`'. They are ASCII text files which may be created and modified with a text editor. Their basic syntax is as follows.

- Blank lines are ignored.
- Lines beginning with '`#`' are considered to be comments and are ignored.
- Lines ending with '\' (backslash) are continued on the next line.
- The description for each database comprises several 'keyword:value' pairs. Each keyword:value pair occupies a single line, unless continued over more than one line. The individual keywords are described in Section 10.1, below. Most are optional.
- The first keyword of each database entry must be `serv_type`.
- Subsequent keywords of each database entry can occur in any order.
- Unrecognised keywords are ignored (to permit future extensions).
- Many of the optional keywords correspond to facilities that are not supported by most databases.
- Some of the keywords require a list of values. All such lists have the same basic format as a Tcl[2] list: usually a list of words or strings enclosed in double-quotes ('`"`') and separated by one or more space characters. Alternatively, curly brackets ('`{ }`') may be used instead of double-quotes.
- If a keyword's value comprises more than one list then these lists are separated by a colon ('`:`').
- If a keyword's value includes a variable reference (see Section 10.6, below) then the variable is expressed in Tcl format: in practice it will start with a dollar character ('`$`').

[2]The Tcl scripting language is described by its author, John Ousterhout, in his *Tcl and the Tk Toolkit*[6].

10.1 Keywords

The entry for an individual database in a configuration file comprises some of the following keywords. The entry for the database must begin with the `serv_type` keyword. Other keywords can occur in any order. The keywords are optional unless otherwise indicated. The keywords are case-sensitive and must be specified entirely in lower case.

`serv_type` (mandatory) The type of database. The options permitted are described in Section 10.2, below.

`long_name` (mandatory) A one-line description of the database. Typically the client will display it to allow the user to identify the database.

`short_name` (mandatory) A short name for the database. Conventionally it has the form:

database`@`*institution*

where *database* is an abbreviation for the database and *institution* a standardised abbreviation for the institution where the on-line version is located. By convention *institution* has three or four characters; some common values are listed in Table 1.

Abbreviation	Institution
`cadc`	Canadian Astronomy Data Centre, Dominion Astrophysical Observatory
`eso`	European Southern Observatory, Garching bei München
`lei`	Department of Physics and Astronomy, University of Leicester
`roe`	Royal Observatory Edinburgh

Table 1: Abbreviations for institutions hosting ACL servers

`url` (mandatory) The URL and query template to access the server. See Section 10.3.

`symbol` Defines how objects in the returned table should be plotted. See Section 10.6.

`copyright` A copyright notice for the client to display.

`search_cols` A list of columns on which range searches are permitted. See Section 10.5.

`sort_cols` The columns on which the returned table is sorted.

`sort_order` The order into which the returned table is sorted. The permitted values are: `increasing` and `descreasing`.

`show_cols` By default a client will display all the columns in the returned table. If a `show_cols` list is specified then by default only the columns in the list will be displayed. Also the order of the list defines the order in which the columns should be displayed.

`id_col` The number of a column in the returned table which contains a unique identifier (or 'name') for each object in the table. By default the first column of a TST format table contains such an identifer (see Section 11).

`ra_col` The number of a column in the returned table which contains the Right Ascension. By default the second column of a TST format table contains the Right Ascension (see Section 11).

`dec_col` The number of a column in the returned table which contains the Declination. By default the third column of a TST format table contains Declination (see Section 11).

`x_col` The number of a column in the returned table which contains x image pixel coordinates.

`y_col` The number of a column in the returned table which contains y image pixel coordinates.

`is_tcs` If this keyword is present it should be set to '1'. Its presence indicates that the returned table either is in or should be converted to TCS format. TCS format catalogues have fixed column names and rules for converting column names when catalogues are imported from other formats.

`help` A URL pointing to a Web page (or pages) describing the database.

`backup1` A reserve URL to be used to access the database in the case where the URL specified by the `url` keyword does not respond.

`backup2` A second reserve URL to be used to access the database in the case where the URLs specified by the `url` and `backup1` keywords do not respond.

For keywords `id_col`, `ra_col`, `dec_col`, `x_col` and `y_col` the number of a column is defined as its sequence number in the list of column names which defines the columns in a TST table (see Section 11 and Figure 2). The first column is numbered zero. This sequence number is sometimes referred to as the 'column index'.

10.2 Types of servers

The types of database which may be specified for keyword `serv_type` are as follows. If an unrecognised type is specified it will be ignored (to permit future extensions).

`catalog` The database is a simple catalogue; this is the simplest and most common option.

`archive` The database is an archive. This option is similar to '`catalog`', but the table returned may contain special columns whose values are URLs giving access to 'bulk data' (images, spectra, time-series *etc*) for the selected objects. See Section 11.1 for details of these special columns.

`namesvr` The database is a name server. The only type of query permitted is to submit to the server a character string corresponding to the name of an astronomical object. The server returns a TST catalogue with a single row and three columns. The row corresponds to the object named and the three columns are: identifier (name), Right Ascension and Declination. See configuration file `simpleconfig.cfg` for an example of a `namesvr` entry.

`imagesvr` The database is an image server. See Section 12.

`local` The database is a local catalogue in the TST format.

`directory` The 'database' is another configuration file. This facility allows a tree (or rather network, since recursion is permitted) of linked configuration files to be built up. The `url` keyword gives the URL of the destination configuration file. The term 'directory' comes from making an analogy with an hierachical file system. See configuration file `simpleconfig.cfg` for an example.

10.3 URL and query specification

The `url` keyword prescribes how the client should access a remote server to query a database. Following the usual conventions for a CGI gateway it consists of the URL corresponding to the script which constitutes the server, followed by a '`?`' and then a string defining the query passed to the server. The query string consists of simple characters and 'tokens'. The tokens start with a '%' character. When a client makes a query the tokens are replaced with values which correspond to the individual query and the resulting string is sent to the server. An example query string for a server which provides circular area searches on a single catalogue might be:

```
ra=%ra&dec=%dec&radius=%r2
```

which contains the tokens `%ra`, `%dec` and `%r2`. The client might substitute them to yield:

```
ra=10:30:00&dec=-30:30.0&radius=3
```

The format of the query string must correspond to that expected by the server. This restriction aside, considerable freedom is allowed in the format of the query string. See configuration file `simpleconfig.cfg` for examples. The complete list of the permitted tokens is as follows.

`%ra` Right Ascension (see Section 10.4).

`%dec` Declination (see Section 10.4).

`%x` x image pixel coordinate (see Section 10.4).

`%y` y image pixel coordinate (see Section 10.4).

`%r` radius of a circular area search (see Section 10.4).

`%r1` inner radius (see Section 10.4).

`%r2` outer radius (see Section 10.4).

`%w` width of a rectangular region (see Section 10.4).

`%h` height of a rectangular region (see Section 10.4).

`%m1` Minimum magnitude.

`%m2` Maximum magnitude.

`%m` Maximum magnitude when no minimum is specified.

`%n` Maximum number of objects (or rows) to return.

`%cols` A list of the columns which the returned table is to contain. If no value is supplied all the columns in the database will be returned. The order of the list corresponds to the order in which the columns are returned.

`%id` The name (or identifier) of an object submitted to a `namesvr` ('name server') database.

`%mime-type` The HTTP mime-type to be inserted in a HTTP `get` command.

`%sort` A list of columns on which the returned table is to be sorted. If no value is specified the table is returned unsorted.

`%sortorder` The order into which the returned table is sorted. The permitted values are `increasing` and `decreasing`.

`%cond` A string specifying one or more range searches. See Section 10.5.

10.4 Coordinates, units and formats

By default databases are assumed to contain columns of Right Ascension and Declination on which they can be searched. However, some databases do not contain such coordinates, but rather have Cartesian x, y positions. Typically such positions occur in catalogues generated by detecting the objects present in a CCD direct image frame or a digitised photographic plate. The following rules apply.

(1) By default databases are assumed to be searchable on columns of Right Ascension and Declination. Further, these columns are assumed to be called '`ra`' and '`dec`', respectively. Simply include the tokens `%ra` and `%dec` in the query.

(2) If a 'positional' search (ie. one finding objects in either a circular area or a region bounded by meridians of Right Ascension and parallels of Declination) is to be made on columns of Right Ascension and Declination then the formats and units of the given values should be:

Right Ascension sexagesimal or decimal hours,

Declination sexagesimal or decimal degrees.

If a sexagesimal value is entered than a colon (':') should be used as a separator. The coordinates should be for equinox and epoch J2000.

(3) If a database is searchable on x, y positions rather than celestial coordinates then the following keywords should be added to its entry in the configuration file:

`ra_col:`	`-1`
`dec_col:`	`-1`
`x_col:`	(sequence number of column holding x coordinate)
`y_col:`	(sequence number of column holding y coordinate)

Setting `ra_col` and `dec_col` to `-1` indicates that columns of Right Ascension and Declination are not available.

(4) The units of the x, y positions are simply 'pixels', that is, dimensionless numbers. Note however, that they are of type REAL rather than INTEGER; positions can be expressed to a fraction of a pixel.

(5) The units in which the radius, height and width of circular and 'rectangular' positional searches are expressed also vary depending on whether the database contains celestial coordinates or x, y positions. The tokens affected are:

```
%r %r1 %r2 %w %h
```

The alternatives for their units are:

in a search on celestial coordinates: decimal minutes of arc,

in a search on x, y positions: decimal pixels.

10.5 Range queries

To permit range queries on one or more columns of a database the keyword `search_cols` should be included in the entry for the database in the configuration file and the `%cond` keyword should be included in the query part of its `url` keyword. These items have the following syntax.

`search_cols` **keyword** A list of one or more column names together with text prompts for the minimum and maximum values of the range required.

`search_cols:` *column-name-1 minimum-prompt-1 maximum-prompt-1* `:`
column-name-2 minimum-prompt-2 maximum-prompt-2 ...

The client uses the minimum and maximum prompts when soliciting the extrema of the required range from the user. An example might be

```
search_cols:  mag "Bright limit" "Faint limit" :  \
b_v "Minimum colour" "Maximum colour"
```

`%cond` **token** This token is included in the query part of the `url` keyword and specifies any range search that is supplied. The syntax of the values substututed into the `%cond` string is:

column-name-1 = *minimum-value-1*`,`*maximum-value-1*`&` *column-name-2* = *minimum-value-2*`,`*maximum-value-2* ...

When a search is specified *column-name-n*, *minimum-value-n* and *maximum-value-n* are substituted with respectively the column name, minimum value and maximum value. The server should process this string and return only rows that lie within the given ranges. An example of substituted query might be:

```
mag=1.0,2.0&b_v=0.1,0.3
```

10.6 Plotting directives

The keyword `symbol` prescribes how objects in the returned table of values are to be displayed in finding charts, image overlays *etc*. Considerable flexibility is allowed in the way that objects are plotted. In particular, the two common types of cases which are conventionally used in atlases and finding charts are supported:

- the size of the plotted symbol is scaled according to some property not directly related to the apparent size of the object. For optical star catalogues the magnitude is traditionally used,
- the size and orientation of the plotted symbol bears some relation to the apparent size and orientation of the image. For example, galaxies are often plotted as ellipses with the outline of the ellipse approximating to some isophote.

The `symbol` keyword only prescribes how objects should be plotted in finding charts and image overlays; it is not appropriate for scatter-plots where the two axes are not celestial coordinates or x, y positions. The syntax of the `symbol` keyword is:

`symbol:` *column-names symbol-info size-expr* `:` ...

This triumavirate of items can be repeated an arbitrary number of times, with each occurence being separated by a colon ('`:`'). The meaning of each of the three items is as follows.

column-names is a list of the names of columns in the returned table which appear as variables in *symbol-info* or *size-expr*. The names should be separated by spaces. Column names containing embedded spaces or characters which Tcl[3] interprets as special should be enclosed in curly brackets ('`{ }` '). Alternatively, names which contain spaces but not any characters which Tcl regards as special may be enclosed in double-quotes ('`" "`').

symbol-info prescribes the appearance of the plotting symbol. See Section 10.6.1 for details.

[3]The Tcl scripting language is described by its author, John Ousterhout, in his *Tcl and the Tk Toolkit*[6].

size-expr is an expression which is evaluated to give the size of each symbol. This expression uses Tcl syntax, with some or all of the column names listed in *column-names* appearing as variables. The expression may optionally be followed by the units of the value computed. The units permitted are:

`image` image pixels,

`deg J2000` degrees for equinox J2000,

`deg B1950` degrees for equinox B1950.

Symbol	Scale and rotate?
`square`	
`circle`	
`triangle`	
`cross`	
`diamond`	
`plus`	●
`ellipse`	●
`compass`	●
`line`	●
`arrow`	●

Table 2: Plotting symbols

Colour	Name
default	(default)
red	`red`
green	`green`
blue	`blue`
cyan	`cyan`
magenta	`magenta`
yellow	`yellow`

The default colour is the opposite of the plot background. Usually it will be black or white.

Table 3: Colours for plot symbols recognised by CURSA

10.6.1 symbol-info

symbol-info prescribes the appearance of the symbol plotted. It is a list comprising the following items:

symbol colour ratio angle label condition

symbol is mandatory; the other items are optional.

symbol is the plotting symbol to be used. The values permitted are listed in Table 2. The symbols marked with a bullet ('•') are typically scaled and rotated to correspond to the appearance of the image; see *ratio* and *angle*, below.

colour is the colour in which the symbol is plotted. It may be any valid X colour name. If *colour* is omitted then GAIA and *SkyCat* draw the symbol in black and white, which stand the best chance of being visible when overlaid on a colour image. CURSA will recognise only the restricted set of colours listed in Table 3.

ratio* and *angle are respectively the width / height ratio of the object and rotation from north. Both quantities may be given as expressions involving column names.

label is the label for each object. It may be fixed text, an expression involving column names or a mixture of the two.

condition is a boolean expression involving column names whose value prescribes whether the object is plotted or not. If the expression evaluates to `true` (or 1) then the object is plotted; if it evaluates to `false` (or 0) it is not. By having a list of entries for the `symbol` keyword, each with a different condition attached to the *symbol-info* item different symbols can be plotted for each object, depending on the values of its column entries.

10.6.2 Examples

`symbol:  "" circle 12`
Plots objects as circles of constant size.

`symbol:  mag circle 15-$mag`
Plots objects as circles scaled according to magnitude (column `mag`).

`symbol:  mag {circle red} 15-$mag`
Plots objects as red circles scaled according to magnitude.

`symbol:  {a b pa} {ellipse red $a/$b $pa} {$a/3600.0 "deg J2000"}`
Plots objects as red ellipses. The ratio of the ellipse is computed from the semi-major and semi-minor axes (`a` and `b` respectively). The orientation is computed from position angle (`pa`). The image size is computed in degrees from the semi-major axis, which is assumed to be in seconds of arc.

```
symbol:  {mag rv} {circle red "" "" "" $rv>0 } 15-$mag : \
     {mag rv} {circle blue "" "" "" $rv<=0 } 15-$mag
```
Plot objects as circles scaled according to magnitude. However, the colour of each circle depends on the radial velocity (`rv`). Objects with a positive radial velocity are shown in red, those with a negative one in blue.

10.6.3 Inserting private Tcl procedures

If a standard Tcl expression does not allow you to calculate the symbol size as you wish then you may be able to invoke your own Tcl procedure to perform the calculation. This facility is only available if you are using *SkyCat* as a client and is only likely to be useful with your own catalogues. An example might be:

```
symbol "rmag bmag" circle "[my_plot_proc $rmag $bmag]"
```

where you provide Tcl procedure `my_plot_proc`.

10.7 Checking a configuration file

A Perl script is available to check the validity of configuration files. You can use it to check for errors in any configuration files that you have created or modified. It is included with the examples provided with this document as file:

```
/star/examples/ssn75/checkcfg
```

You can either run it from the examples directory or copy it to a convenient local directory. On non-Starlink systems you may need to edit the first line to correspond to wherever Perl is located on your system.

To use the script simply specify the name of the configuration file, optionally preceded by the appropriate directory specification, on the command line:

```
    /star/examples/ssn75/checkcfg  configuration-file-name
```

For example, to check a copy of the example configuration file supplied with this document (assuming that there is a copy in your current directory):

```
    /star/examples/ssn75/checkcfg   simpleconfig.cfg
```

If the configuration file is valid the `checkcfg` will display the message:

```
    Configuration file parsed successfully.
```

Conversely, if there are problems with the configuration file then explanatory error messages are displayed. Usually the number of the invalid line in the configuration file is included in the message. Note that `checkcfg` is mostly checking for syntax errors; it does not, for example, check that any URLs specified are valid.

11 The Tab-Separated Table Format

This section gives a brief description of the tab-separated table (TST) format. ACL servers should return the list of selected objects in this format. Various packages, including GAIA, CURSA and Starbase can also read local files containing catalogues in this format. There are alternative descriptions of it in SUN/214[4], the Starbase FAQ[4] and the *Astronomical Catalogue Library User Manual*[2].

TST format files are text files. They are usually generated by a remote server in response to a query from a local client. However, they could equally well be local files created with a text editor. Figure 2 shows a simple example of a tab-separated table. This example is available as file:

```
    /star/examples/ssn75/simple.TAB
```

The description of the table and the table of values occupy the same file and the description occurs at the start of the file. Most of the description shown in Figure 2 is optional.

The first line of the description is a title. Lines beginning with a '`#`' are comments which are ignored.

Parameter definitions start with the parameter name, and a colon ('`:`') is appended to the end of the name to identify it as such. The rest of the line contains the parameter value. The name and value are the only information stored for each parameter. The example contains two parameters: `EQUINOX` and `EPOCH`.

Any remaining lines in the description (apart from the last two, which immediately precede the table of values) are free text.

The only mandatory items in the description are the two lines immediately before the table of values. The first of these lines is the list of column names. Each name is separated by a single tab character (ASCII code nine; strictly speaking the horizontal tab). In the figure tab characters are shown as '`<tab>`';

[4]http://cfa-www.harvard.edu/ john/starbase/FAQ.html

```
Simple TST example; stellar photometry catalogue.

A.C. Davenhall (Edinburgh) 26/7/00.

Catalogue of U,B,V colours.
UBV photometry from Mount Pumpkin Observatory,
see Sage, Rosemary and Thyme (1988).

# Start of parameter definitions.
EQUINOX: J2000.0
EPOCH: J1996.35
# End of parameter definitions.

#column-units: <tab>Hours <tab>Degrees <tab>Magnitudes <tab>Magnitudes <tab>Magnitudes
#column-types: CHAR*6 <tab>DOUBLE <tab>DOUBLE <tab>REAL <tab>REAL <tab>REAL
#column-formats: A6 <tab>D13.6 <tab>D13.6 <tab>F6.2 <tab>F6.2 <tab>F6.2

Id<tab>ra<tab>dec<tab>V<tab>B_V<tab>U_B
--<tab>--<tab>---<tab>-<tab>---<tab>---
Obj. 1<tab> 5:09:08.7<tab> -8:45:15<tab>  4.27<tab>  -0.19<tab>  -0.90
Obj. 2<tab> 5:07:50.9<tab> -5:05:11<tab>  2.79<tab>  +0.13<tab>  +0.10
Obj. 3<tab> 5:01:26.3<tab> -7:10:26<tab>  4.81<tab>  -0.19<tab>  -0.74
Obj. 4<tab> 5:17:36.3<tab> -6:50:40<tab>  3.60<tab>  -0.11<tab>  -0.47
...
```

Figure 2: A simple tab-separated table. Note that in a tab-separated table the list of column names, sequences of dashes and fields in the table are separated by tab characters. In this figure tab characters are indicated by '`<tab>`'. (Note that the first column, `Id`, is an object name and hence its units are left blank. Thus, the CURSA-specific `column-units:` is separated from the following `<tab>` character by only one or more spaces; see Section 11.3 for details of the CURSA extensions.)

obviously a real tab-separated table would contain actual tab characters instead. The name is the only mandatory information stored for each column. The TST format places few restrictions on the column names: they can contain spaces, special and punctuation characters *etc*. However, it is usually a prudent precaution to restrict column names to contain only alphanumeric and underscore ('`_`') characters and to make the first character alphabetic. If these precautions are observed then fewer problems are likely to occur if the table is subsequently converted to another format or read by a variety of different clients. Remember that if you are writing a server which returns a TST table via the Internet then you do not know which client will be used to access it.

The line immediately after the list of column names indicates the end of the description and the start of the table. It consists solely of dashes and tab characters. By convention there are as many sequences of dashes as there are column names, each sequence is separated by a single tab character and each has the same number of dashes as there are characters in the corresponding column name.

In the table of values each row occupies a single line. Individual fields are separated by a single tab character. The fields occur in the same order as the corresponding column names.

11.1 Special columns of identifiers and celestial coordinates

The TST format has the following additional rules for special columns containing identifiers and celestial coordinates.

(1) By default, the first three columns in the table are: an identifier (that is, an object name), Right Ascension and Declination. These columns are usually called `Id`, `ra` and `dec` respectively.

(2) If the first three columns of the table are not an identifier, Right Ascension and Declination then the TST should contain the following three parameters:

```
id_col
ra_col
dec_col
```

The value of each parameter should be the number of the appropriate column or, if there is no such column, `-1`. The column number is the sequence number of the column in the list of column names, starting counting at zero.

(3) The Right Ascension and the Declination are either both in decimal degrees or the Right Ascension is in sexagesimal hours and the Declination is in sexagesimal degrees. If sexagesimal notation is used then a colon ('`:`') is used as the sexagesimal separator.

(4) The equinox and epoch are specified using parameters `equinox` and `epoch` respectively. The default value if the equinox and epoch are not specified is J2000.0.

11.2 Conventions for tables returned by ACL servers

In addition to the basic TST format the following additional conventions apply to tables returned by ACL servers.

(1) Keywords specified for a database in its entry in the configuration file are equivalent to, and treated as, parameters in the TST. Thus, `id_col`, ra_col and `dec_col` are usually supplied as keywords in the configuration file rather than parameters in the TST returned by the server. If a client writes a table of retrieved objects as a local file in the TST format it should include these keywords as parameters.

(2) Some types of database, particularly `archives` (see Section 10.2), may return tables containing columns called `more` and `preview`. These columns are optional. However, if present they should be used as follows.

`more` a URL giving more information on the object. The destination URL will typically be a conventional page of HTML, suitable for display with a Web browser.

`preview` a URL pointing to an image of the object.

11.3 CURSA extensions

Some additional items can be added to the TST header information which provide additional information about the catalogue and allow CURSA[3] to interpret it more precisely. These items begin with a '`#`' character and thus are TST comments. Consequently a catalogue which contains them remains perfectly standard and valid and can be processed with software other than CURSA. The items are illustrated in Tables 1 and 2 and are as follows.

`#column-units:` is followed by a tab-separated list of units for the columns.

`#column-types:` is followed by a tab-separated list of data types for the columns. The permitted data types are listed in Table 4.

`#column-formats:` is followed by a tab-separated list of display formats for the columns. Fortran-like, FITS-compatible formats are used. This convention facilitates converting tables between the TST and FITS tables formats without loss of information.

CURSA Data Type	Description	Standard Fortran 77?
BYTE	Signed byte	No
WORD	Signed word	No
INTEGER	Signed integer	Yes
REAL	Single precision	Yes
DOUBLE	Double precision	Yes
LOGICAL	Logical	Yes
CHAR[$*n$]	Character string	Yes

n is the number of elements in the character string; it is a positive integer.

Table 4: CURSA data types

In all cases the items are listed in the order in which they occur in the table. There is no tab character between `#column-units:`, `#column-types:` and `#column-formats:` and the following value. Values in the special TST columns of Right Ascension and Declination (that is, those identified by the `ra_col` and `dec_col` parameters) are always interpretted using the TST rules for representing angles rather than the CURSA ones.

11.4 MIME type

The first line of information returned by an ACL server should always be the MIME[5] type:

[5]Multipurpose Internet Mail Extensions. The MIME protocol was originally developed to allow non-text data to be included in e-mail messages and was subsequently adopted for use with HTTP.

```
Content-type:  text/plain
```

This line is strictly speaking not part of the TST table and will be used by the client or Web browser to interpret the format of the data which follow.

12 Image Servers

Image servers differ from other types of ACL server in that they return a direct image or 'pixel map' of a region of sky rather than a catalogue of objects. The image is returned formatted as a FITS[6] file which may optionally be compressed.

(1) Image servers have a configuration file entry similar to those for other types of database. The server type is `imagesvr`. The region of sky required is specified by giving the central Right Ascension and Declination (or x, y pixel position if celestial coordinates are not available), width and height. The usual tokens (see Section 10.3):

```
%ra  %dec  %x  %y  %w  %h
```

are included in a query string as part of a `url` keyword in the usual fashion. The tokens have their usual meaning and units. An example configuration file entry for an image server is:

```
serv_type:      imagesvr
long_name:      Digitized Sky at ESO
short_name:     dss@eso
url:            http://archive.eso.org/dss/dss\
?ra=%ra&dec=%dec&mime-type=%mime-type&x=%w&y=%h
copyright:      Digitized Sky Survey (c) by AURA, provided online by ESO
```

(2) The server parses the query and generates a FITS file corresponding to the region specified. It first returns the MIME type, followed by the FITS file which optionally may be compressed. The MIME types corresponding to the various types of compression are listed in Table 5. For example, the MIME type for a gzipped FITS file is:

```
Content-type:  image/x-gfits
```

Acknowledgements

The material in Part II is largely based on Section 2 of Allan Brighton's *Astronomical Catalogue Library User Manual*[2]. Martin Bly, Allan Brighton, Peter Draper and Mike Read provided useful comments on the draft version of the document. Any mistakes, of course, are my own.

[6]http://fits.gsfc.nasa.gov/fits_home.html

MIME type	Type of compression
`image/fits`	uncompressed
`image/x-hfits`	H compressed
`image/x-gfits`	gzipped
`image/x-gstarbase`	gzipped (alternative)
`image/x-cfits`	UNIX compressed
`image/x-cstarbase`	UNIX compressed (alternative)
`text/html`	error message in HTML format

Table 5: MIME types for compressed FITS images

Bibliography

[1] M. Albrecht, M. Barylak, D. Durand, P. Fernique, A. Micol, F. Ochsenbein, F. Pasian, B. Pirenne, D. Ponz and M. Wenger, 19 September 1996, *Astronomical Server URL* (Version 1.0). See URL: `http://vizier.u-strasbg.fr/doc/asu.html` 2

[2] A. Brighton, 16 January 1998, *Astronomical Catalog Library User Manual*, issue 3.1 (document number GEN-SPE-ESO-19400-0949), European Southern Observatory Very Large Telescope Data Management Division. 2, 9, 11, 12

[3] A.C. Davenhall, 25 July 2000, SUN/190.8: *CURSA — Catalogue and Table Manipulation Applications*, Starlink. 1, 6.4, 11.3

[4] P.W. Draper and N. Gray, 27 January 2000, SUN/214.7: *GAIA — Graphical Astronomy and Image Analysis Tool*, Starlink. 1, 11

[5] I.S. Graham, *The HTML Sourcebook*, 1995 (John Wiley and Sons: New York). 2

[6] J.K.Ousterhout, 1994, *Tcl and the Tk Toolkit* (Addison-Wesley: Reading, Massachusetts). 2, 3

[7] R.L. Schwartz, 1993, *Learning Perl* (O'Reilly and Associates Inc: Sebastopol, California). 2

[8] L. Wall and R. L. Schwartz, 1991, *Programming Perl*, (O'Reilly and Associates Inc: Sebastopol, California). 2