# CSDS User Interface
# ISDAT Architectural Design Document

Swedish Institute of Space Physics
Uppsala Division
S-755 91 Uppsala
Sweden

with change bars for versions 2.0 and 2.1

| Document Status Sheet | | | |
|---|---|---|---|
| 1. Document Title: **CSDS-UI ISDAT Architectural Design** | | | |
| 2. Document Reference Number: **DS-IRF-AD-0001** | | | |
| 3. Issue | 4. Revision | 5. Date | 6. Reason for Change |
| Draft | 0 | 94 Oct 13 | New document. Delivered to ESRIN 14 Oct. |
| 0 | 1 | 94 Nov 2 | Section 4.3 added, and section 4.2 modified accordingly. Moved section 2.3 (Terminology) to the Introduction. Included Dblib manual pages as Appendix ??. Added Islib manual pages as Appendix ??. Added Isutillib manual pages as Appendix ??. Added more details to the *cuitm* description section ??. Added more details to the *CSDS module* description, section ??. Added more details to the *cuimeta* description, section ??. Added more details to the *ISDAT kernel* description, section ??. |
| 0 | 2 | 94 Nov 25 | Specified the joining algorithm, section ??, page ??. |
| 1 | 0 | 95 Mar 04 | Changes related to RID:s of version 0.1. RID ds-est-rid-001: Authors removed from the title page and from the reference list on page ??. RID ds-est-rid-002: Included reference to environment settings in section ??. RID ds-est-rid-003: Figure ?? has been corrected. RID ds-est-rid-004: Figure text of Figure ?? has been corrected. RID ds-est-rid-005: Added more detailed info on memory and disk space in section ??, page ??. RID ds-est-rid-006: Corrected UR.30 and UR.31 entries of table ?? RID ds-est-rid-007: Added a description of the handling of several concurrent users in section ??. RID ds-est-rid-008: Clarified that component 2.2.2 hard copy also handles ASCII flat files (section ??. RID ds-est-rid-009: Added information about the ISDAT configurability in section ??. RID ds-ral-rid-0001: Figure ?? has been modified to include "User ID" as input to the kernel. RID ds-ral-rid-0002: Removed the two figures of section ??. Referring to the overall architectural design document instead. Indicated the possible extension to a local server. RID ds-ral-rid-0003: A more homogeneous description of "Type" is used. In most cases "executable" and "process" have been replaced by "module" in section ??. RID ds-ral-rid-0004: Clarified the interpretation of the block diagrams in section ??. RID ds-esr-rid-0001: Included a reference to the internal interface document whereever an environment variable is mentioned. Added client 2.7 cuistat to Figure ??. Added the whole section ??. In section ??, it is explained that the manual pages, included as appendices, are not guaranteed to be up to date. Added a 0 in the document ID to conform with other CSDS documents. |
| | | | |

Document Status Sheet

1. Document Title: **CSDS-UI ISDAT Architectural Design**

2. Document Reference Number: **DS-IRF-AD-0001**

| 3. Issue | 4. Revision | 5. Date | 6. Reason for Change |
|---|---|---|---|
| 1 | 1 | 95 Mar 09 | Changed module numbers 2.2.10 - 2.2.14 in Figure **??**.. |
| 2 | 0 | 95 Aug 29 | Modifications associated with CSDS User Interface, Release 4 (ISDAT 2.2). The section **??** is updated and also the figures **??**, **??** and **??**. The component "host/user validation" in section **??** is updated. The components "Registry", "Query", "GetContent", "GetData", "map file administration" and "authorization" in section **??** are updated. The component ID 1.2.4 in section **??** is not used. The component "server search" is added to section **??** and the figures **??** and **??** are updated. The component "configuration file load and save" is added to the sections **??** and **??**, and the figures **??** and **??** are updated. |
| 2 | 1 | 95 Oct 14 | A paragraph concerning version numbers is added to the components "Get Content" and "Get Data" in section **??** and to the component "client start-up" in section **??**. The component "Get Data" in section **??** is updated. |
| | | | |

# Contents

# 1 Introduction

## 1.1 Purpose of the document

This document describes the architectural design of the CSDS User interface ISDAT modules. It is intended for use within the CSDS UI software development teams at IRF-U, ESRIN, ESTEC, and RAL.

## 1.2 Scope of the software

The scope of the software is to provide tools for data manipulation and display of the CSDS data bases.

## 1.3 Definitions, acronyms, and abbreviations

The used acronyms and abbreviations are explained in Table ??.

| Acronym | Meaning |
|---|---|
| AD | Applicable Document |
| ANSI | American Standard Code for Information Interchange |
| CDF | Common Data Format |
| CD-ROM | Compact Disc Read Only Memory |
| CPU | Central Processing Unit |
| CSDS | Cluster Science data System |
| CUI | CSDS User Interface |
| DBH | Data Base Handler |
| ESA | European Space Agency |
| ESRIN | European Space Research INstitute |
| ESTEC | European Space Technology Centre |
| GUI | Graphical User Interface |
| IACG | International Agency Coordination Group |
| ID | Identification |
| IDL | Interactive Data Language |
| IRF-U | Institutet för Rymdfysik, Uppsalaavdelningen |
| | Swedish Inst. of Space Phys., Uppsala Division |
| ISDAT | Interactive Science Data Analysis Tool |
| ISTP | International Solar Terrestrial Programme |
| LAN | Local Area Network |
| N/A | Not Applicable |
| NDC | National Data Centre |
| OSF | Open Software Foundation |
| PPD | Prime Parameter Data |
| PPDB | Prime Parameter Data Base |
| RAL | Rutherford Appleton Laboratory |
| R2 | Release 2 |
| SPD | Summary Parameter Data |
| SPDB | Summary Parameter Data Base |
| TBD | To be defined |
| TBW | To be written |
| TCP/IP | Transmission Control Protocol / Internet Protocol |
| UI | User Interface |
| UR | User Requirement |
| WAN | Wide Area Network |
| X11R5 | X11 Release 5 |

Table 1: Acronyms and abbrieviations

## 1.4 Terminology

In this document, we will use the notation *ISDAT* when we discuss the basic ISDAT structure. The notation *CSDS UI (CUI) Data Manipulation package* will be used when the customised ISDAT package to be incorporated in the overall CSDS user interface is referred to. The CSDS UI Data Manipulation package consists of two parts, an *ISDAT server*, corresponding to the *Data Base Handler, DBH* in ISDAT terminology, and an *ISDAT client package* corresponding to a selection of *ISDAT clients* in ISDAT terminology.

## 1.5 Applicable documents

Applicable documents are:

**AD1** CSDS-UI Overall Architectural Design document [Ref. ?].

**AD2** CSDS-UI User Requirements Document [Ref. ?].

**AD3** CSDS-UI Internal Interface Document [Ref. ?].

## 1.6 Overview of the document

The general structure of the document adheres to the ESA recommendations for the architectural design phase [Ref. ?]. Some on-line manual pages are included as appendices in this document for the convienience of the reader. They may not be up to date at all times. The reader is referred to the on-line pages for the updated version.

# 2 System overview

## 2.1 CSDS User Interface

The CSDS UI Data Manipulation Package constitutes one component of the CSDS User Interface. The CSDS User Interface is described in **AD1**.

## 2.2 The basic ISDAT structure

The CSDS UI Data Manipulation package is built on an already existing structure, *ISDAT*. The core of ISDAT consists of a well defined, project independent, interface between a data base handler, DBH (server), and a scientific analysis and display software package (clients), and a mechanism for communication between the data base handler and the clients. ISDAT utilises a client/server model, implying a full flexibility regarding physical locations of analysis programs and data bases. The use of the ISDAT in the CSDS User Interface is described in the Overall Architectural Document [Ref. ?] . The nominal implementation is to have an ISDAT server at tha National Data Centres and ISDAT clients at the scientific user's platforms. . It is also possible to run ISDAT servers locally at the user's platforms.

# 3 System context

The CUI Data Manipulation system context is shown in Figure ?? (see section ?? page ?? for explanation of the symbols).

The interfaces are described in the following sections.

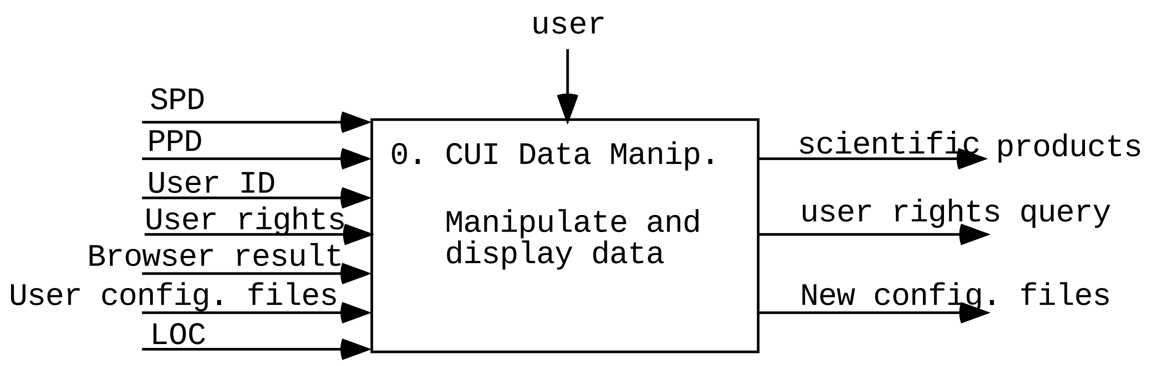

Figure 1: CUI Data Manipulation system context

## 3.1 Data Input

The input to the system consist of external data bases and information from the ESRIN server and client used for access control.

**PPD** The Prime Parameter Data consist of spin averaged data (about 4 seconds) from all four satellites. The content of the PPDB is described in [Ref. ?]. Parts of or the complete data base is stored in CDF format at the National Data Centres. The storage format is described in [Ref. ?].

**SPD** The Summary Parameter Data consist of one minute averaged data from one satellite. The content of the SPDB is described in [Ref. ?]. Parts of or the complete data base is stored in CDF format at the National Data Centres. The storage format is described in [Ref. ?].

**LOC** CSDS CDF files residing at the local workstation.

**User ID** The user ID and an associated code number is provided by the ESRIN client software at start of the ISDAT client package. The format is described in **AD3**.

**User rights** A list of user rights is provided by the ESRIN server upon request by the ISDAT server. The format is described in **AD3**. For PP & SP and for LOC files a local access rights is used.

**Browser result** An ASCII file describing the result of a catalogue browser session. The file is created by the ESRIN client software. The format is described in **AD3**.

**User configuration files** These are files describing the user preferences on the client platform regarding graphics formats etc.

## 3.2 Control data flow

**user** The CUI Data Manipulation package consist of an interactive software controlled by the end user (scientist) normally working at a personal work station in contact with an with an ISDAT server at national data centre.

## 3.3 Data output

The output from the CUI Data Manipulation module is:

**scientific products** This is the final scientific product that may take a multitude of formats, for example a screen graph, a postscript file, CDF file, or a flat file.

**user rights query** This is a CUI internal product that eventually result in a user rights list as data input (see **AD3**).

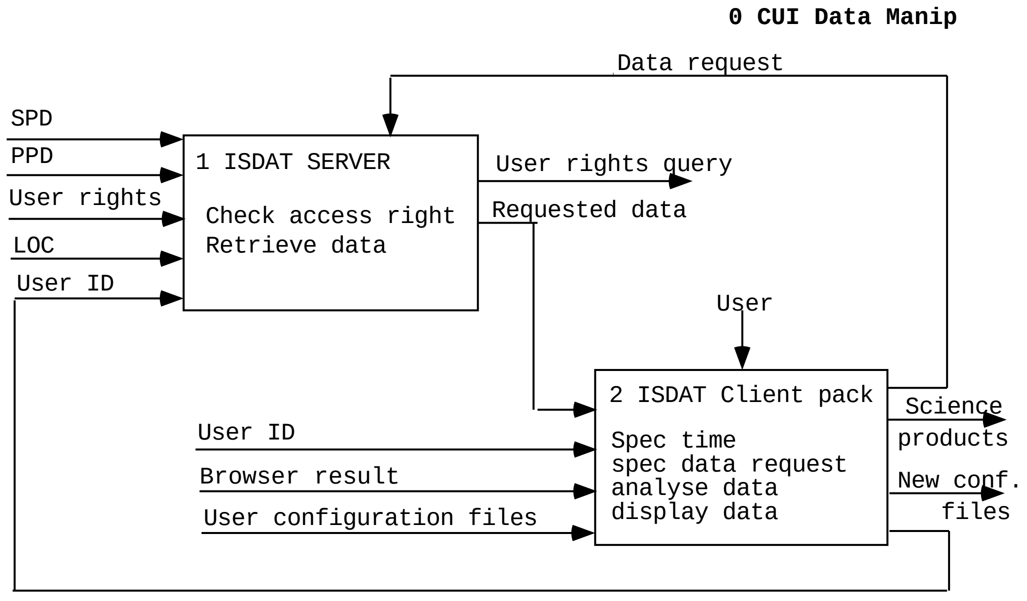

Figure 2: CUI Data Manipulation data flow

**New configuration files** Graphics definition files that the user wishes to save.

## 4 System design

### 4.1 Design method

The CUI Data manipulation package is based on re-use and modification of existing software. Therefore, no rigorous analysis method could be used for the architectural design.

The high level design is presented in terms of data flow diagram in a form influenced by the structured analysis method. Boxes represent functions (activity described by a verb). Arrows from the left, into the boxes, represent input data, arrows to the right, out of the boxes, represent output data, and arrows from above represent control data. One level of decomposition may be described by one or several diagrams. The level is indicated by a successive expansion of the numbering levels.

Since a substantial fraction of the software modules consist of libraries, the lower level design is described in terms of block diagrams instead of data flow diagrams. The block diagrams indicate the level of a particular module as well as the interdependencies between modules. High level modules depend on modules vertically below the module. The modules can also interact horisontally. The globally used libraries *Dblib, Islib* and *Isutillib* are described in detail in Appendices ??, ??, and ??.

### 4.2 Decomposition description

The top level data flow diagram is shown in Figure ??.

The two top level components are described in section ??.

The interface between the two main components is of client/server type. That is, the two processes (ISDAT server and ISDAT clients) are individual processes that may or may not run on the same workstation and at the same geographical location.

The interfaces external to the CUI Data manipulation package are described in section ??. The internal interfaces are described in section ??.

### 4.3 Internal Interfaces

The well defined internal server/client interface is in fact the key to the project independent nature of the ISDAT. There are three internal data interfaces between the ISDAT server and the ISDAT client package (see Figure **??**):

**Data request** The *data request* control data flow may be an actual data request or a query of the available instruments and their properties. The data request allows for a very flexible specification of the data. The data request is specified in a complex data structure *desc* specified in great detail in [Ref. ?]. The actual call to request data is described in the on-line manual for call *DbGetData*, see Appendix **??** page **??**..

**Requested data** The data is returned in a likewise complex and general data structure described in the on-line manual for call *DbGetData*. See also [Ref. ?].

**User ID** The *User ID* is simply a forwarding of the User name and session key ($CUI_USERNAME and $CUI_SESSION_KEY) (see [Ref. ?]) to the client (see section **??** page **??**. The information is forwarded at connection setup .

## 5 Component description

Note that the following subsection numbers correspond to the module numbers if the leading section (**??**) number is removed.

### 5.1 ISDAT SERVER

The original ISDAT Server accommodates the unpacking and calibration of experiment data, as well as formatting of the data communicated to the clients requesting the data. The original ISDAT server is designed to randomly access the data, and is capable of handling a wide variety of requests in a general manner. Examples of such capabilities are data gap handling, interpolations, delivery of raw or calibrated data, supplying alphanumeric strings corresponding to delivered units, signals etc., warning flags and messages. The ISDAT server is also capable of responding to client queries regarding instrument descriptions and available data hierarchy at the connected particular server. The available data is specified in terms of *conceptual instruments* that may or may not directly correspond to the actual hardware instruments. The conceptual instrument is described in a hierarchical manner as *project-member-instrument-sensor-signal-channel-parameter*. The ISDAT original server can handle multidimensiona l data.

The detailed descriptions are given in the on line manuals. In the CUI Data Manipulation package, only one data base specific module, the *CSDS module*, is supplied.

Clients can connect locally or remotely using TCP/IP protocol. Several data base handlers can run simultaneously on one workstation. The DBH is built in a modular structure, with all project specific software residing in separate modules. The local installation thus only includes one or several project modules of use for that particular installation.

The decomposition of the CUI ISDAT server is shown in the data flow diagram of Figure **??**.

To enable the ISDAT server to handle many requests at the same time a concurrent server mechanism has been developed.

The first instance of the server called the parent server will never process any requests itself. When a connection request arrives the parent server forks a copy of itself called the child server. The child server inherits the connection from the parent server and will handle all future request on this connection. The parent server will close the connection and will be immediately ready to accept a new connection request.

When a child server is created it will change its port number on where to listen for connections, it will pass back that port number to the client that caused the creation of the child server. The client will

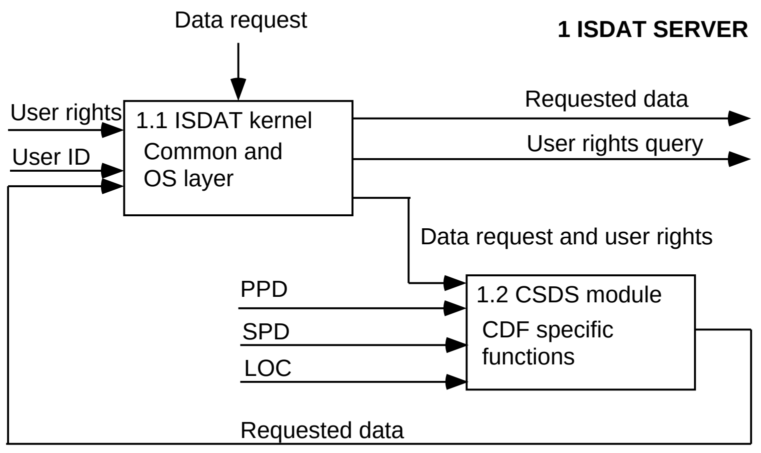

Figure 3: CUI ISDAT SERVER data flow

take this port number and change its ISDAT_DATABASE (see [Ref. ?]) environment variable which means that all new clients started from this client will use the just created child server. When all the connections to a child server are closed it will kill itself.

In the normal case the user will start a time manager and then start all other clients and extra time managers from it, the concurrent server mechanism will in that case allocate one server for exclusive use by this user.

The change to the ISDAT_DATABASE (see [Ref. ?]) environment variable is calculated by taking the database number of the parent server and multiply it by 256 (MAXSERVERS value), adding 100 and adding the instance number given to the child server.

```
Example 1:
The parent server is started as dbh :0.
The users environments defines ISDAT_DATABASE=host:0.
The first users time manager will propagate ISDAT_DATABASE=host:100
to all new clients and time managers started from it.
The second users time manager will propagate ISDAT_DATABASE=host:101
to all new clients and time managers started from it.
The second users time manager will propagate ISDAT_DATABASE=host:102
to all new clients and time managers started from it.
```

```
Example 2:
The parent server is started as dbh :2.
The users environments defines ISDAT_DATABASE=host:2.
The first users time manager will propagate ISDAT_DATABASE=host:612
to all new clients and time managers started from it.
The second users time manager will propagate ISDAT_DATABASE=host:613
to all new clients and time managers started from it.
The second users time manager will propagate ISDAT_DATABASE=host:614
to all new clients and time managers started from it.
```

```
The number of client servers can be limited by changing the variable
common.serverLimit in the isdat.server configuration file.
Setting it to zero will disable the concurrent server mechanism
which will make the parent server to handle all requests in an
iterative manner.
There is a related variable called common.clientLimit which will
limit the number of connections per server.
```

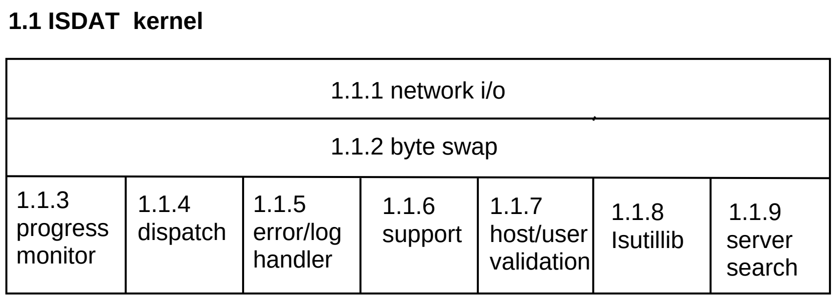

Figure 4: CUI ISDAT Kernel block diagram

### 5.1.1 ISDAT kernel

The *ISDAT kernel* receives the requests from the clients, validates it and forwards the request to the relevant module. In the CUI application only one module is available, the CSDS module. The decomposition of the ISDAT kernel is shown in Figure **??**.

The *CUI ISDAT Kernel* consist of:

**Component ID 1.1.1 network i/o**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03

**Function** The network interface is based on sockets and uses the stream based TCP/IP and Unix domain protocols. When the server is started a socket used to listen for new connections is created for each supported protocol by calling CreateWellKnownSockets(). After that the dispatcher will be in control. When the dispatcher is ready for a new task it will call WaitForSomething() which will block until a new request or a new connection arrives. If a new connection arrives the new connection will be accepted and set up in EstablishNewConnections(). To function ReadRequestFromClient() is available to read a complete request from the client. The function WriteToClient() is available to write arbitrary data to the client, the write is buffered and the buffer can be flushed by the function FlushAllOutput().

**Subordinates** see Figure **??**

**Dependencies** None

**Interfaces** see Figure **??**

**Data** see Figure **??**

**Component ID 1.1.2 byte swap**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03

**Function** Performs the byte swapping necessary to allow the client and server to run on machines with different byte order. The two different byte orders are little-endian (e.g.. DEC) and big-endian (e.g.. Sun). Conversions between different floating point standards are not performed, IEEE-754 is assumed. Byte swap is only performed when the representation of the server and the client differs and is handled at the server side.

Swap of requests are handled in the file swapreq.c where each client request has its own swap function, referenced from the SwappedProcVector[] in tables.c. Swap of replies are handled in the file swaprep.c where each client reply has its own swap function, referenced from the ReplySwapVector[] in tables.c. There are also swap functions defined for each the data structures that is passed as part of some replies. Swap of events is also handled in swaprep.c where each event has its own swap function referenced from the EventSwapVector[] in tables.c. Currently only two events are defined; error event and progress event.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Component ID 1.1.3 progress monitor**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.45i

**Function** To notify the client about the amount of work remaining for a time consuming request. An instrument module can call the function ProgressEvent(ClientPtr client, int procent) at any time to hint the client of how much work is done. The percent argument must be set to the percentage of the work done. ProgressEvent() will generate an asynchronous event to the client.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.1.4 dispatch**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03

**Function** Inspects the incoming request and translates it into a call to the relevant function (e.g.. GetData, GetContent, Query, ...). It inspects the data specification hierarchy and decides which instrument module that needs to be called.

When the database handler is started control will be passed to Dispatch() which will enter an infinite loop waiting for new requests and connections. Each request (service) has a corresponding function in the dispatcher:

**ProcBadRequest()** Called if an unknown request arrives.

**ProcGetSync()** Sends back a dummy reply. Can be used by a client to verify that the server is alive.

**ProcControl()** Allows a control message to be sent to an instrument module. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbControl.3 man page in Appendix **??** page **??**.

**ProcGetData()** Called when the client requests data. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbGetData.3 man page in Appendix **??** page **??**.

**ProcGetContent()** Called when the client requests information about available on-line data. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbGetContent.3 man page in Appendix **??** page **??**.

**ProcGetInfo()** Called when the client requests static information about a data specification hierarchy. Typical information is coordinate parameters and sensor types. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbGetInfo.3 man page in Appendix **??** page **??**.

**ProcChangeHosts()**

**ProcListHosts()**

**ProcPrepareData()** Currently not used.

**ProcQuery()** Called when the client requests information about available data specification hierarchies. It uses the registry mechanism (FindFunction()) to find all available instrument modules, it will call them one by one and merge the results. For details on the service see the DbQuery.3 man page in Appendix **??** page **??**.

**ProcOverview()** Called when the client requests a detailed view of what data is available for a specified time period. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbOverview.3 man page in Appendix **??** page **??**.

**ProcDownload()** Allows a client to download an arbitrary block of data to an instrument module. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbDownload.3 man page in Appendix **??** page **??**.

**ProcUpload()** Allows a client to upload an arbitrary block of data from an instrument module. It uses the registry mechanism (FindFunction()) to decide which instrument module should handle the request. For details on the service see the DbUpload.3 man page in Appendix **??** page **??**.

**ProcInitialConnection()**

**ProcEstablishConnection()** Used to pass connect information between the client and server when the connection is established. Byte order information is exchanged and the user name and authorisation values are passed to the server.

There is also a mechanism to notify an instrument module of a client connection and to call an instrument module at regular 1 minute intervals.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.1.5 error/log handler**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.10c

**Function** The log handler is initialised by a call to LogHandle *LogInit(char *path, char *module) where path is the name of the log file to use and module is the top level module name. The module name for all kernel errors will be "sys" and the module name for the CSDS module will be *Csds*. The path for the kernel is $CUI_USR_ROOT/log/dbh_sys.err (see [Ref. ?]) and the path for the Csds module is $CUI_USR_ROOT/log/dbh_csds.err (see [Ref. ?]) . A log message is logged by calling void LogError(handle, client, action, message) which is implemented as a macro that adds the ANSI symbols __FILE__ and __LINE__ to identify the file name and line number in the source file. Arguments:

**handle** log handle from LogInit()

**client** client handle

**action** LOG_ERROR to log an error, LOG_FATAL to log an error and terminate the server

**message** the error message string

Example:

```
LogError(sysLog, 0, LOG_ERROR, "bad something");
will produce the entry:
Oct 25 11:37:52 130.238.30.13 al Csds Query.c 22: bad something
in the log file. Explanation of the fields:
```

|  |  |
|---|---|
| `Oct 25 11:37:52` | `time stamp` |
| `130.238.30.13` | `IP-address of connecting client` |
| `al` | `user name on client side` |
| `Csds` | `module name` |
| `Query.c` | `file name` |
| `22` | `line number` |
| `bad something` | `error message` |

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.1.6 support**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.32b

**Function** Various support functions of which the most noticeable ones are:

**GrabAllFileDescriptors()** Some Unix implementations allows the maximum number of file descriptors to be increased up to system defined hard limit. This function will perform that task.

**ProcessCommandLine()** Parses the dbh command line options. The supported options are:

|  |  |
|---|---|
| `:<no>[.<base>]` | `the server number given by <no> and the base port number given by <base>` |
| `-help` | `to get list of available options, default is 0.14734` |
| `-ld <kbytes>` | `limit data size to number of <kbytes>` |
| `-ls <kbytes>` | `limit stack size to number of <kbytes>` |
| `-pn` | `partial network. allows the server to run even if it fails to use one protocol family one protocol` |
| `-to <sec>` | `number of seconds within a connection is required to complete, if not it will be killed, default is 60 seconds` |

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??** This component is interfacing the ESRIN software package, see **AD3**.

**Component ID 1.1.7 host/user validation**

**Type** module

**Purpose** Provides access control when the server is used for CSDS local files access. UR.42b

**Function** At server startup the access control file given by the *.common.hosts line in the server configuration file isdat.server is read. If the file doesn't exist or the *.common.hosts line is not found the access control is disabled which means that the server allows any connections. The file consists of lines of the form:

```
# comment
<host name>
<host name>     <user1>, <user2>
```

The first form will allow all users on machine <host name> to connect. The second form will allow access to user <user1> and <user2> on machine <host name> to connect. Example file:

```
# ISDAT local server access control file
# allow user eric and nils on irfu
irfu    eric, nils
# allow all users on abba
abba
```

Users on the local machine is always allowed to connect.

The module defines the functions:

InitHosts()
Called at server startup to intitialize the module and read the access control file.

InvalidHostAndUser()
Called at each client connection to verify the access rights for the connecting user.

GetHosts()
Protocoll interface entry point that allows a client to get a list of all allowed hosts and users.

AddHost()
Protocoll interface entry point that allows a client to add hosts to the access control list. This function is only executed if the requesting client is on the local host.

RemoveHost()
Protocoll interface entry point that allows a client to remove hosts from the access control list. This function is only executed if the requesting client is on the local host.

ChangeAccessControl()
Protocoll interface entry point that allows a client to enable/disable the access control mechanism. This function is only executed if the requesting client is on the local host.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.1.8 Isutillib**

**Type** Library

**Purpose** To fulfil user requirements (see **AD2**): UR.03

**Function** Contains various support functions needed by the server and all clients. The bulk of the module deals with conversions between various representations of time and the internal time used by the ISDAT interfaces. It also defines a number of include files that tries to hide machine dependency from the other modules. The individual functions of Isutillib are described in Appendix **??**.

**Subordinates** see Figure **??**, **??**, **??**, **??**, **??**, **??**, **??**

**Dependencies** see Figure **??**, **??**, **??**, **??**, **??**, **??**, **??**

**Interfaces** see Figure **??**, **??**, **??**, **??**, **??**, **??**, **??**

**Component ID 1.1.9 server search**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.15, UR.46

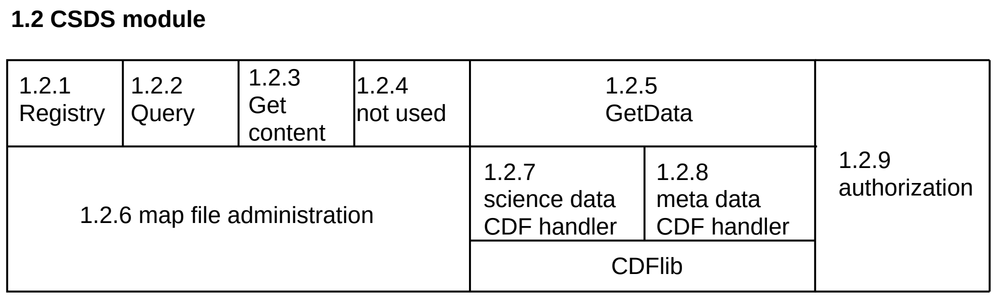

Figure 5: CUI ISDAT CSDS module block diagram

**Function** Implements the Search service and will return the data intervals matching the search criteria. A time interval is given as well as conditions on some quantities. These conditions can contain arithmetical and logical expressions. Requests, for each quantity, are then sent to respective database. The returned data is checked and a list of time intervals, for which the conditions are fulfilled, are set up. Time intervals that are shorter than the given integration time are rejected from the list. Then time intervals, of those that remain, which are closer than the given time resolution are concatenated. The resulting list of time periods are returned.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

### 5.1.2 CSDS module

This is the module handling the CDF specific processing of the CSDS data, and the only data base specific server module supplied with the CUI Data Manipulation Package. The decomposition is shown in Figure **??**.

The *CUI ISDAT CSDS module* consist of:

**Component ID 1.2.1 Registry**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.46

**Function** During server start-up each instrument module will be requested to identify itself and to register all the services it implements.

The Csds module registers the following services for the csds_pp and csds_sp projects:

**GetContent** request for information on available on-line data

**GetData** request for data

**Query** request for list of available data specification hierarchy

**Control** control message for closing all CDF files (DbCONTROL_RELEASE) and map file generation (DbCONTROL_GENERATE) used by cuimgen

**Timer** calls module at 1 minute intervals, used to close CDF files after RELEASE_AGE minutes since last access. RELEASE_AGE is currently set to 20 minutes

**Connect** called when a client connects to set up authorisation state

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.2 Query**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.46

**Function** Implements the Query service and will respond with a data specification hierarchy. The relevant levels in this hierarchy are: project, member, instrument and sensor.

- Available projects are: CSDS_PP, CSDS_SP and CSDS_LOC.
- Available members are: C1, C2, C3, C4, CL, PM, SM and HM.
- Available instruments are: ASP, AUX, CIS, DWP, EDI, EFW, FGM, PEA, RAP, STA, WHI and MIX.
- Available sensors (variables) depends on the values of the previous levels and are dynamically assigned.

For each combination of project (database), member (spacecraft) and instrument there is an associated quantity file. Such a file contains the variables found in the associated CDF files. The content of each such file is placed into the associated Map structure. So here the names of the variables will be taken from this Map structure.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.3 Get content**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.46

**Function** Implements the GetContent service and will respond with a list of all available on-line data.

Get content gives a list of the time intervals where data is available for a certain combination of database, instrument and spacecraft.

The dataVersion will affect what versions of the CDF files to use. If set to the constant DbUNKNOWN then the files with the highest version numbers will be picked out. If set to a number, only the files with that version number will be picked out.

The first thing that will happen in GetContent is that all map files and quantity files will be read in to each Map structure. The pointers to these map structures will then be stored in a global array (csdsMapTable in GetContent.c). This array is also used in Get data and Query.

After that, the Map structure that corresponds to the given combination of database, instrument and spacecraft will be located. From this Map structure it's possible to get a list of the time intervals where we have continuous data. This list will be very long and is therefore compressed in the following way: if the difference between the stop time of one interval and the start time of the next interval is less than INTERVAL_DIFF_LIMIT seconds, then the two intervals will be concatenated to one interval. INTERVAL_DIFF_LIMIT is currently set to 7200 seconds.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.4** Not used.

**Component ID 1.2.5 GetData**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.46

**Function** Implements the GetData service and will return the requested data.

Get data returns a list of data segments for the requested time interval. Each segment contains continuous data.

The dataVersion will affect what versions of the CDF files to use. If set to the constant DbUNKNOWN then the files with the highest version numbers will be picked out. If set to a number, only the files with that version number will be picked out.

The first step is to locate the time intervals within the requested time interval where the requested variable is continuous. This locating process is done in Locate.c. The first step in the locating process is to get the Map structure that corresponds to the given combination of database, instrument and spacecraft. From this Map it's possible to construct a list of the CDF's covering the requested interval. To be in this list the CDF must fulfil the following requirements:

- The CDF has data in the requested interval
- The CDF is not fully covered by earlier CDF:s in the list
- The CDF contains the requested variable
- The version of the CDF must be the latest or the explicit desired

For each CDF in this list we will also save the time, from which the CDF is unique. Then there will be a check for each of these CDF's if it is in the list of open CDF's. The name of this list is orbitList (see Locate.c). An open CDF is kept in an Orbit structure (see Orbit.h). orbitList contains the latest used CDF's. The reason to have such a list is to minimise the number of open/close CDF. If the CDF is open, then the address of its Orbit structure will be placed into the next free element of the array orbitPtrList (see Locate.c). If the CDF is not open, then it will be opened and an Orbit structure will be created. The Orbit structure will be placed in orbitList and its address will be placed in orbitPtrlist. When we have checked the CDF's, then orbitPtrList points to the CDF's covering the requested interval.

Now it's time to construct the time intervals in which the given variable is continuous. This will be done for each CDF in the orbitPtrList. To do that we will use the Map structure mentioned above. From it we can get the following information for each CDF:

- the time intervals in which the variable may be continuous.
- the time intervals where the variable has gaps.

From this information it's possible to construct the time intervals where the variable is continuous. These intervals will then be saved in the Orbit structure under the member name varIntervals. Having these intervals, it is possible to construct the intervals lying within the interval, where the CDF is unique(not covered by other CDF:s). These intervals, unique for this CDF, will be saved in the Orbit structure under the member name usedVarIntervals.

So now we have located the time intervals within the requested time interval where the requested variable is continous. The only thing that now remains is to sort the data segments in ascending order according to the start time (of the data segment).

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.6 map file administration**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.46

**Function** Due to the number of CDF files involved the overhead of trying to keep track of all the CDF files internally is to large. The map files hold some basic information like data intervals and data gaps to allow rapid lookup of the requested data. One map file is kept per database, instrument and spacecraft. A related file type is the quantity file which hold information about the available variable names. One quantity file is kept per database, instrument and spacecraft.

To update a map file it will be read in to a Map structure (see Map.h). Initially there is no map file to read in and then just a Map structure will be created. A Map structure contains among other things pointers to the following areas: the Header area, the Entry area, the Interval area and the Gap area, the Variable area and the String area. The Header area keeps track of the number of units in the other areas.

The Entry area contains one MapEntry structure (see Map.h) per CDF. A MapEntry structure contains information about:

- the name of the CDF (or a pointer to the name).
- where in the Interval area the continuous data intervals of the CDF can be found and how many they are.
- where in the Gap area the gaps of the CDF can be found and how many they are.
- where in the Variable area the variables of the CDF can be found and how many they are.

The Interval area contains one MapInterval structure (see Map.h) per continuous data interval. The information about the continuous data intervals is taken from the global attribute Data_intervals in the CDF file. Note! Continuous data interval has the following meaning here: an interval is continuous if data exists for any variable at each successive time.

The Gap area contains one MapGap structure (see Map.h) per gap. The MapGap structure keeps information of which variable that has a gap and where the gap starts and stops. The information of where a variable has gaps is given from a check in the CDF file where this variable contains fill values.

The Variable area contains the variable sets for the different CDF files.

When the map file has been read in, the date of each CDF file in the current directory will be compared to the date of the map file. If the CDF file is older than the map file then nothing will happen, because in this case the CDF file is already registered in the map file. But if the CDF file is newer than the map file, the CDF file will be opened and the information mentioned above will be read from it. When all CDF files in the current directory has been checked, the Entry area will be sorted with regard to date. After that, the relevant parts of the Map structure will be written to the map file, which then is updated.

In parallel with the updating of the map file the corresponding quantity file will be updated as well. For each CDF file newer than the map file its variable names will be compared to the ones already in the quantity file. If the CDF file contains some new variable names they will be added to the quantity file.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.7 science data CDF handler**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.27, UR.28, UR.29, UR.30, UR.31[1] , UR.46

**Function** Handles science data requests and merging over midnight. It uses the map/quantity file facility to rapidly look up the requested data. A cache of open CDF's is maintained for the last recently used files thus avoiding the overhead of open/close of CDF files for repeated accesses.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.8 meta data CDF handler**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.28, UR.29, UR.30, UR.46

**Function** Handles meta data requests. Both global and variable attributes are handled. Merging over midnight is supported.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.2.9 authorization**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.03, UR.46, UR.08b

**Function** Implements the interface to the access control library. Any access violation will be reported to the user as a bad access error. The actual retrieval of access information will be performed at client connect to reduce the number of queries to the Oracle server.

The resulting privileges are stored in the ppAccess and spAccess fields of the AccessState structure pointed to by the modulePrivate field of the client structure. This allows each service module to rapidly verify if the client is authorised or not.

The PP access may be restricted to a campaign access mode which will restrict a users to a set of instruments and data intervals. The list as queried from the ACA oracle interface is converted to an internal format pointed to by campaign fiels of the AccessState structure. The function CsdsCampaignCheck() is called by each service module to verify the PP campaign access rights.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**. This component is interfacing the ESRIN software package, see **AD3**.

## 5.2 CUI ISDAT Client package

By *ISDAT clients* in general, we understand analysis and display programs by which the user interacts with the ISDAT and receives his products. There are three classes of clients:

1. time managers
2. general clients

[1]Data will be joined into a common time line using the nearest sample as joining algorithm. The data set with the smallest sampling rate should be the governing data set

3. specific clients

A particular client may or may not be a part of the particular ISDAT installation. It could well be a personal client residing only at one local workstation. Every client is self standing program (main program) that may or may not run at the same work station as the ISDAT server from which it gets its data. The two latter classes of clients may include direct links to commercial program packages like IDL. The data flow related to clients is shown in Figure **??** where the activity *2 ISDAT client package* can be replaced by one specific client.

*Time managers* are special *clients* that are used to coordinate the behaviour of an associated family of clients. A typical set of functions for a *time manager* may be:

**Activate other clients** At start time, the time manager traverses the ISDAT directory tree, identifies all executable files, builds a menu or a list of clients. The user then can select clients from the list to be activated and added to the group of clients controlled by the particular time manager. A particular client can also be started from a command line. However, in that case, servers with access control cannot be used.

**Select project** At start time the time manager identifies available *project, member, and instrument* (see **??**), and the user may choose one of the available combinations.

**Select data file** At start time the time manager identifies available data periods, and the user can select a suitable time interval. As the time interval is updated, all active clients, in the family, are informed about the currently selected interval. This means that the user can have several graphic windows open and he can be sure that all windows belonging to the family of clients represent the same time interval.

Several time managers, each controlling its family of clients, may be active simultaneously. In the CUI distribution one time manager *cuitm* is included. Several parallel processes of the *cuitm* can be active concurrently however.

The *general clients* do not depend on any particular project or instrument etc. It normally starts by queering the ISDAT server about the supported data bases and its properties and build up menus to support the user in requesting data.

The *specific clients* depend on a particular project or instrument etc. and are intended for particular data analysis purposes.

The users ISDAT environment is configurable by means of a set of configuration files and a set of environment variables that can be set at run time or be included in the users login files.

The decomposition of the CUI ISDAT client package is shown in Figure **??**.

### 5.2.1 cuitm

*cuitm* is the CUI client of class *time managers* (see section **??** above). The structure of the *cuitm* is illustrated in Figure **??**.

The *cuitm* consist of the following components:

**Component ID 2.1.1 cuitmGUI**

**Type** task

**Purpose** To fulfil user requirements (see **AD2**): UR.43a, UR.53, UR.54, UR.33a (alphanumeric input), UR.33b (alphanumeric input), UR.35, UR.39, UR.40

**Function** A graphical event driven user interface. XDesigner will be used as the design tool.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces**
- Input to *all registered clients* module: new time message is broadcasted.

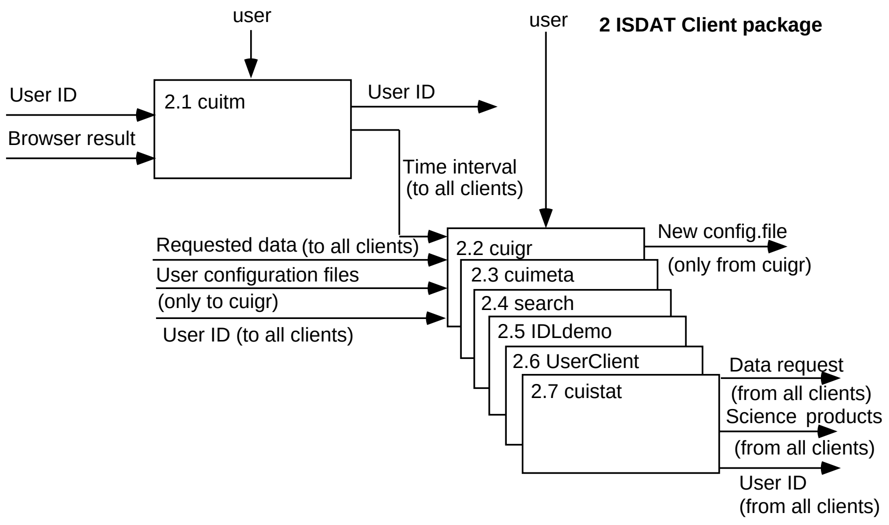

Figure 6: CUI ISDAT client package data flow

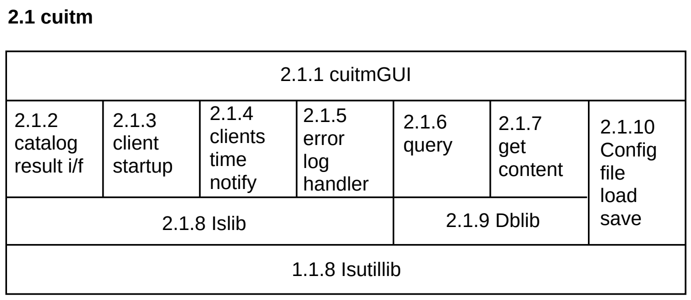

Figure 7: CUI ISDAT cuitm block diagram

- Output from *catalog result i/f* module: New catalog description structure.
- Output from *client startup* module: List of all clients that could be started within a time manager session.
- Output from *query* module: A three level tree of available projects, members, and instruments hierarchy. Then *cuiGUI module* recreate new projectmenuW, memberMenuW, and instrumentMenuW global Widgets.
- Output from *get content* module; Available on-line data list for selected project, member, instrument.

**Resources** Display running an X server.

**Reference** [Ref. ?]

**Data in:** Global structure cuitmList with catDesc and contentList elements.

**in:** Global structure cuitmSpec (project, members, and instruments hierarchy)

**in:** Global structure cuitmGlobal (database name, argc, argv,...)

**out;** Global structure cuitmTime

**Component ID 2.1.2 catalog result i/f**

**Type** task

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** Read and parses the query_result.res (see [Ref. ?]) written by the catalogue browser. The file is checked for update once per second.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??** Input to *cuiGUI* module; If the *query_result.file* (see [Ref. ?]) state has changed, an appropriate signal is sent.

**Data** Out: Global structure catDesc with all information from *query_result.res* (see [Ref. ?]) file or if the file disappeared, a CAT_NOCATALOG parameter.

**Component ID 2.1.3 client start-up**

**Type** task

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** Reads the bin directory and looks for files with the extension .cl, the file contains a description of the client. It builds a list of available clients, creates a hierarchical menu and starts the client upon request from the GUI module. Check connection to an ISDAT server.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces**
- Input to all modules: Initialise cuitmGlobal structure (data base name,...),
- Input to *cuiGUI* module; Initialise client menu, initial time and start spec. tree.
- Output from *query*; Get spec. tree (project, member, instrument)

**Resources** Display running an X-server. ISDAT server.

**Data out:** Global structure cuitmSpec ( project, member, instrument tree ) and initial projectMenuW, memberMenuW, instrumentMenuW global Widgets.

**out:** Global structure cuiTime (initial time),

**out:** Global cilentMenuW Widget.

**Component ID 2.1.4 clients time notify**

**Type** task

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** When the user changes time/interval a message is sent to all clients attached to the time manager, this enables clients to work with a common time. The message mechanism used is based on X properties and events.

The dataVersion number is part of the time message and the data version selected in the time manager is thus passed to all clients attached to the time manager. Each data manipulation client will then pass the dataVersion on each request to the database handler (server).

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Resources** Display running an X-server.

**Component ID 2.1.5 log handler**

**Type** task

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** Will log abnormal errors to the log file. User error will be communicated to the user through error dialogues.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces**
- Input to *client startup*: Create initial spec.tree (project, member, instrument) hierarchy.
- Input to *cuiGUI*: Recreate spec.tree (project, member, instrument) hierarchy.

**Component ID 2.1.6 query**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** The time manager has no hardwired knowledge of the data specification hierarchy. It will use the Query service to ask the database handler (server) about which available projects, members and instruments that are available.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Data** Out: Global structure cuitmSpec (project, member, instrument tree).

**Component ID 2.1.7 get content**

**Type** task

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** Uses the GetContent service to present a list of available on-line data intervals for the selected project, member and instrument.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??** This component is interfacing the NDC data bases, see [Ref. ?]. Input to *cuiGUI* module: Fill a content list to be displayed.

**Resources** ISDAT server.

**Data** Out: Global structure cuitmList ( contentList field).

**Component ID 2.1.8 Islib**

**Type** library

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** Library implementing the communication protocol between time managers and clients. It also provides support for creating dynamic menus. The individual functions of *Islib* are described in Appendix **??**.

**Subordinates** see Figure **??**, **??**, **??**, **??**, **??**, **??**

**Dependencies** see Figure **??**, **??**, **??**, **??**, **??**, **??**

**Interfaces** see Figure **??**, **??**, **??**, **??**, **??**, **??**. Input to all other modules except *Dblib* ( provides miscellaneous functions )

**Data** Numerous structures.

**Component ID 2.1.9 Dblib**

**Type** library

**Purpose** To fulfil user requirements (see **AD2**): UR.35

**Function** Library used by all clients that provide a c-binding interface to all the services provided by the database handler. The individual functions of *Dblib* are described in Appendix **??**.

**Subordinates** see Figure **??**, **??**, **??**, **??**, **??**, **??**

**Dependencies** see Figure **??**, **??**, **??**, **??**, **??**, **??**

**Interfaces** see Figure **??**, **??**, **??**, **??**, **??**, **??** This component is interfacing the ESRIN software package, see **AD3**.

- Input to *query* module ( use DbQuery function provided in Dblib, see Appendix **??** ) send appropriate structure with new spec.tree.
- Input to *get content* module (use DbGetContent function provided in Dblib, see Appendix **??**) send structure with content list.

**References** A comprehensive explanation of the data structure used is given in [Ref. ?].

**Component ID 2.1.10 configuration file load and save**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36e

**Function** Allows the user to produce an ASCII file saving the complete state of the program, the file can later be loaded to restore all the settings.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component 1.1.8 Isutillib** See section **??**

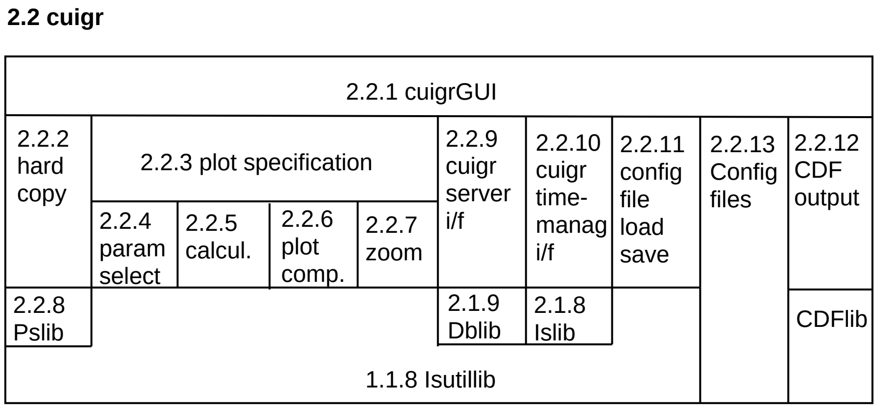

Figure 8: CUI ISDAT cuigr block diagram

### 5.2.2 cuigr

*cuigr* is an ISDAT client of class *general clients* (see section **??** above). It is a general purpose data display and manipulation client. The structure of the *cuigr* is illustrated in Figure **??**.

The *cuigr* client consist of the following components:

**Component ID 2.2.1 cuigrGUI**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.33a (cursor on screen), UR.33b (cursor on screen), UR.36a, UR.39, UR.40

**Function** A graphical event driven user interface. The interface is based on existing design using Motif and Xt intrinsics.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.2 hard copy**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.34, UR.36d

**Function** Produces a postscript or ASCII flat file output of the selected panel(s). The output can be directed to a printer or a file.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.3 plot specification**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** Allows the user to interactively build the plot layout and define its contents.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.4 param select**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** To select the sensor, signal, channel and parameter parts of the data specification hierarchy, it will be presented as a pull down menu. The default project, member and instrument provided by the time manager at client start-up can be overridden in a similar pull down menu.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.5 calculator**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.33c, UR.33d, UR.36g

**Function** Allows the user to bind selected quantities to plots using arithmetic operations. It is implemented in two sections, the upper section and the lower section. The upper section presents the user with a calculator dialogue where all the operations is specified. The lower section is used when plots are (re)generated and performs computations based on the selections in the upper section.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.6 plot composition**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36a, UR.36b, UR.36g

**Function** Allows the user to interactively build the panel layout.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.7 zoom**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** A region of the time axis can be selected and the marked time/interval will be propagated to the time manager which in turn will notify all its clients about the new time/interval.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.8 Pslib**

**Type** Library of module es

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** Postscript library.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.9 cuigr server i/f**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** Interface to Dblib functions (e.g.. DbOpen(), DbGetData() and DbQuery()).

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.10 cuigr time manager i/f**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** Waits for messages from the time manager and will regenerate all plots if a message arrives. After all actions are completed a done message is sent to the time manager.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.11 configuration file load and save**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36e

**Function** Allows the user to produce an ASCII file saving the complete state of the program, the file can later be loaded to restore all the settings.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.12 CDF output**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.32

**Function** Will produce a flat file of the science data, a flat file to CDF converter program will then be started to generate the CDF file.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.2.13 Configuration files**

**Type** ASCII files

**Purpose** To fulfil user requirements (see **AD2**): UR.36f

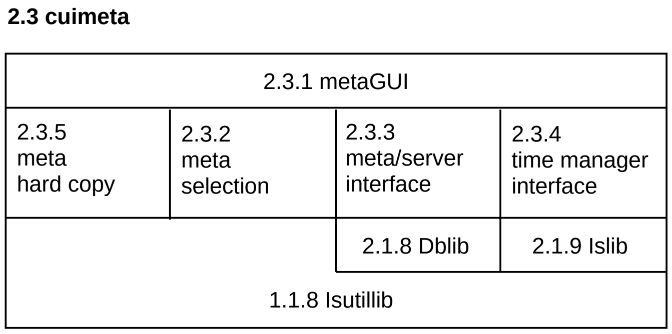

Figure 9: CUI ISDAT cuimeta client block diagram

**Function** A set of pre-defined configuration files describing the set-up to get frequently used summary plots.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.1.9 Dblib** See section **??**.

**Component ID 2.1.8 Islib** See section **??**

**Component ID 1.1.8 Isutillib** See section **??**

**Component ID CDFlib** External CDF libraries. See [Ref. **?**].

### 5.2.3 cuimeta

*cuimeta* is the client that displays the meta data content of the CDF file. The decomposition of the *cuimeta* client is shown in Figure **??**.

The cuimeta client consist of the following components:

**Component ID 2.3.1 metaGUI**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36c, UR.39

**Function** A graphical event driven user interface. XDesigner will be used as the design tool.

**Subordinates** see Figure **??**

**Dependencies** Requires a running time manager and an ISDAT server.

**Interfaces** see Figure **??**

**Component ID 2.3.2 meta selection**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36c

**Function** To produce an ASCII output to a printer or a file.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

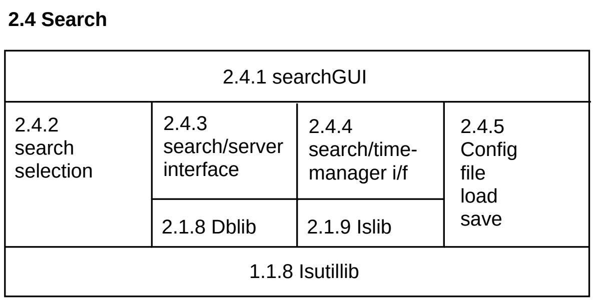
2.4 Search
2.4.1 searchGUI
2.4.2 search selection
2.4.3 search/server interface
2.4.4 search/time-manager i/f
2.4.5 Config file load save
2.1.8 Dblib
2.1.9 Islib
1.1.8 Isutillib

Figure 10: CUI ISDAT search client block diagram

**Component ID 2.3.3 meta server interface**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36c

**Function** Fetch meta data from the server, i.e. an interface to Dblib functions.

**Subordinates** see Figure **??**

**Dependencies** An ISDAT server must be running.

**Interfaces** see Figure **??**

**Component ID 2.3.4 time manager interface**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36c

**Function** Waits for messages from the time manager and will regenerate the meta display if a message arrives. After all actions are completed a done message is sent to the time manager.

**Subordinates** see Figure **??**

**Dependencies** A running server.

**Interfaces** see Figure **??**

**Component ID 2.3.5 meta hard copy**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** Produces an ASCII output of the selected data.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.1.9 Dblib** See section **??**.

**Component ID 2.1.8 Islib** See section **??**

**Component ID 1.1.8 Isutillib** See section **??**

### 5.2.4 search

The *search* client is capable of searching in the scientific data base according to specified search criteria. The search client decomposition is shown in Figure **??**.

The *search* client consist of the following components:

**Component ID 2.4.1 searchGUI**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.15, UR.39, UR.45i, UR.46

**Function** A graphical event driven user interface. XDesigner will be used as the design tool.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.4.2 search selection**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.15, UR.45i, UR.46

**Function** To provide the user with an interactive way to specify the search criteria.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.4.3 search/server interface**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.15, UR.45i, UR.46

**Function** Interface to Dblib functions (e.g.. DbSearch(), DbGetData() and DbQuery()).

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.4.4 search/time manager interface**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.15, UR.45i, UR.46

**Function** Waits for messages from the time manager and will update the relevant parts of the search criteria selection if a message arrives. After all actions are completed a done message is sent to the time manager.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.4.5 configuration file load and save**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.36e

**Function** Allows the user to produce an ASCII file saving the complete state of the program, the file can later be loaded to restore all the settings.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

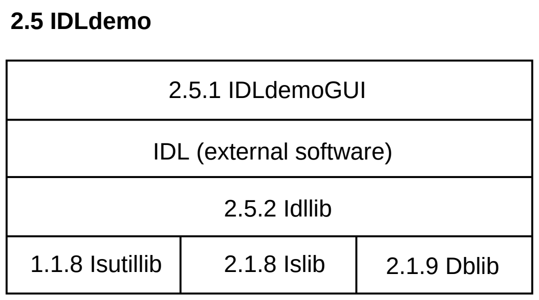

Figure 11: CUI ISDAT IDLdemo client block diagram

**Interfaces** see Figure **??**

**Component ID 2.1.9 Dblib** See section **??**.

**Component ID 2.1.8 Islib** See section **??**

**Component ID 1.1.8 Isutillib** See section **??**

### 5.2.5 IDLdemo

The sole purpose of the *IDLdemo* client is to illustrate to the user how to use the IDL interface to the ISDAT package. There are no requirements of real functionality of the *IDLdemo* client. The *IDLdemo* client decomposition is shown in Figure **??**.

The *IDLdemo* client consist of the following components:

**Component ID 2.5.1 IDLdemoGUI**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.39, UR.44

**Function** Main program written in IDL.

**Subordinates** see Figure **??**

**Component ID IDL** External software package

**Component ID 2.5.2 Idllib**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.44

**Function** A shared library which is loaded into IDL to provide an interface to the time manager and database handler.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 1.1.8 Isutillib** See section **??**

**Component ID 2.1.8 Islib** See section **??**

**Component ID 2.1.9 Dblib** See section **??**.

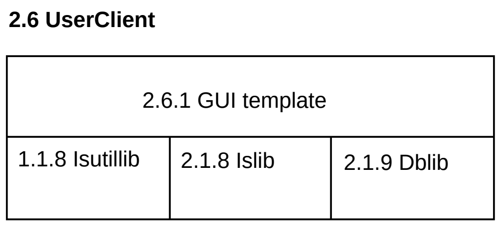

Figure 12: CUI ISDAT userClient block diagram

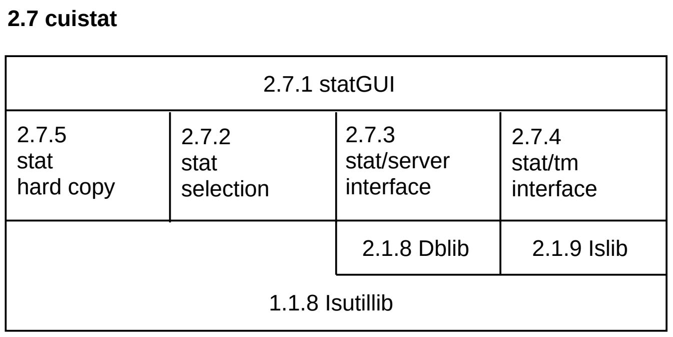

Figure 13: CUI ISDAT cuistat client block diagram

#### 5.2.6 UserClient

The sole purpose of the *UserClient* is to demonstrate to the user how the user can add personal customised clients to the ISDAT package. There are no requirements of real functionality on the *UserClient*. The UserClient client decomposition is shown in Figure **??**.

The *UserClient* consist of the following components:

**Component ID 2.6.1 GUI template**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): UR.35, UR.39

**Function** A demo interface developed using XDesigner.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.1.9 Dblib** See section **??**.

**Component ID 2.1.8 Islib** See section **??**

**Component ID 1.1.8 Isutillib** See section **??**

#### 5.2.7 cuistat

*cuistat* is the client that displays the status data for a particular instrument The decomposition of the *cuistat* client is shown in Figure **??**.

The cuistat client consist of the following components:

**Component ID 2.7.1 statGUI**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): No particular UR.

**Function** A graphical event driven user interface. XDesigner will be used as the design tool.

**Subordinates** see Figure **??**

**Dependencies** Requires a running time manager and an ISDAT server.

**Interfaces** see Figure **??**

**Component ID 2.7.2 stat selection**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): No particular UR.

**Function** To produce an ASCII output to a printer or a file.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.7.3 stat server interface**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): No particular UR.

**Function** Fetch status data from the server, i.e. an interface to Dblib functions.

**Subordinates** see Figure **??**

**Dependencies** An ISDAT server must be running.

**Interfaces** see Figure **??**

**Component ID 2.7.4 time manager interface**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): No particular UR.

**Function** Waits for messages from the time manager and will regenerate the stat display if a message arrives. After all actions are completed a done message is sent to the time manager.

**Subordinates** see Figure **??**

**Dependencies** A running server.

**Interfaces** see Figure **??**

**Component ID 2.7.5 stat hard copy**

**Type** module

**Purpose** To fulfil user requirements (see **AD2**): N/A

**Function** Produces an ASCII output of the selected data.

**Subordinates** see Figure **??**

**Dependencies** see Figure **??**

**Interfaces** see Figure **??**

**Component ID 2.1.9 Dblib** See section **??**.

**Component ID 2.1.8 Islib** See section **??**

**Component ID 1.1.8 Isutillib** See section **??**

# 6 Feasibility and Resource Estimates

The resources for the complete CSDS UI are given in [Ref. ?]. In the following two sections we give the estimated resources for the ISDAT part of the CSDS UI.

## 6.1 CUI ISDAT Server

For the CUI ISDAT server the following resource recommendations are made:

**Disk space** For the installation, 30 MBytes are required for source code and binary files. At run time, an additional 5 MBytes are needed for map files, 5 MBytes for log files, and about 20 MBytes as spare disk space. On the average 200 bytes per CDF file is required for map files. In total a minimum of 60 MBytes is recommended to accommodate the CUI ISDAT Server package.

**Primary memory** About 10 MBytes should be available to avoid degrading of the performance due to swapping. Each user require about 0.5 MBytes of virtual memory in the idle state.

**CPU** more than 50 SPECint92 is recommended.

## 6.2 CUI ISDAT Client Package

For the CUI ISDAT Client Package, the following resources are recommended:

**Disk space** About 50 MBytes is needed for source code and binary files. For user configuration files, 1 MBytes per user is recommended.

**Primary memory** 32 MBytes is recommended to achieve a reasonable performance.

**CPU** More than 25 SPECint92 is recommended.

# 7 Traceability matrix

The full description of the user requirements are given in **AD2**. The capability requirements traceability matrix is given in Table **??**. The constraint requirements traceability matrix is given in Table **??**.

| Functional user requirement | | Component | | Remarks |
|---|---|---|---|---|
| *Catalogue and file management:* | | | | |
| UR.03 | Operation independent of NDC activities | 1 | ISDAT SERVER | |
| *Access control:* | | | | |
| UR.08b | Campaign based access | 1.2.9 | authorization | I |
| *Logging:* | | | | |
| UR.10c | Log user traffic | 1.1.5 | error/log handler | |
| *Query:* | | | | |
| UR.15 | Search files according to science queries | 1.1.9<br>2.4 | CDFsearch<br>search | |
| *Datafile manipulation:* | | | | |
| UR.27 | Subsetting CDF files in time interval | 1.2.7 | science data CDF handler | |
| UR.28 | Subsetting CDF files on variables and search criteria | 1.2.7<br>1.2.8 | science data CDF handler<br>meta data CDF handler | |
| UR.29 | Merge CDF files across midnight | 1.2.7<br>1.2.8 | science data CDF handler<br>meta data CDF handler | |
| UR.30 | Not used | | | I |
| UR.31 | Join CDF files onto common time line using algorithms | 2.2 | igr | I |
| UR.32 | Results of data manipulation to be written as a new CDF file | 2.2.13 | CDF output | |
| UR.32b | It shall be possible to limit the number of concurrent users with respect to data file manipulation | 1.1.6 | support | |
| *Data manipulation* | | | | |
| UR.33a | Interactive time interval selection | 2.1.1 | cuitmGUI | Alphanumeric input |
| | | 2.2.1 | cuigrGUI | Cursor on screen |
| UR.33b | Retrieval request of PPD based on SPD | 2.1.1 | cuitmGUI | Alphanumeric input |
| | | 2.2.1 | cuigrGUI | Cursor on screen |
| UR.33c | Coordinate transformations | 2.2.5 | calculator | |
| UR.33d | Arithmetic operations | 2.2.5 | calculator | |
| UR.34 | Save as flat files | 2.2.2 | hard copy | |
| UR.35 | Interface for user defined modules | 2.6 | UserClient | Not included in R2 |
| | | 2.1 | cuitm | For inclusion of UserClient in menu |
| UR.35b | Limit number of concurrent users with respect to UR.33 | | | UR not applicable with current design. Requirement is covered by UR.32b |
| *Graphic display:* | | | | |
| UR.36a | Plot parameter vs. time | 2.2.1<br>2.2.6 | cuigrGUI<br>plot composition | |
| UR.36b | Support multipanel plots | 2.2.6 | plot composition | |
| UR.36c | Meta data req. | 2.3<br>2.2<br>1.2.8 | cuimeta<br>cuigr<br>meta data CDF handler | Displaying of non-Cluster ISTP, IACG meta data is meaningful only with a local server. |
| UR.36d | Write postscript for hard copy | 2.2.2 | hard copy | |
| UR.36e | User exchange of plot designs | 2.2.12 | config file load and save | |
| UR.36f | Pre-defined set of basic parameters | 2.2.14 | Configuration files | |
| UR.36g | Plot derived parameters vs. time or other parameter | 2.2.6 | plot composition | Not included in R2 |

Table 2: Traceability matrix for the capability requirements

| Functional user requirement | | Component | | Remarks |
|---|---|---|---|---|
| *Human computer interaction:* | | | | |
| UR.39 | Client with GUI at user sites | 2 | ISDAT Client package | |
| UR.40 | Provide X-windows GUI | 2 | ISDAT Client Package | |
| *Architecture and environment:* | | | | |
| UR.42b | Local server | 1.1.7 | host user/validation | I |
| | | 1.2 | CSDS module | I |
| *Architecture and environment:* | | | | |
| UR.43a | SUN Solaris 2.3 or higher | 0 | CUI Data Manipulation | |
| UR.44 | Interface towards IDL | 2.5 | IDLdemo | |
| | | 2.5.2 | Idllib | |
| *Performance:* | | | | |
| UR.45a | Immediate reaction | 2 | ISDAT Client package | |
| UR.45g | GUI and display small computational burdon on NDC platforms | 0 | CUI Data Manipulation | |
| UR.45i | Status and progress info | 1.1.3 | progress monitor | Not included in R2 |
| | | 2.4 | search | |
| | | 2 | ISDAT Client Package | |
| *File formats:* | | | | |
| UR.46 | Interface to CDF files | 1.2 | CDF module | non-Cluster ISTP/CSDS support is meaningless without a local server. |
| *System usability:* | | | | |
| UR.50 | User configurable system | 2 | ISDAT Client Package | |
| UR.53 | Identify non-portable code | 2 | ISDAT Client Package | |
| *Portability:* | | | | |
| UR.54 | Standards: ANSI C, X11R5, OSF/Motif ver. 1.2 | 0 | CUI Data manipulation | |

Table 3: Traceability matrix for the constraint requirements

# A Dblib manual pages

```
NAME
     DbAddEventHandler - adds event handler function

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     Database *DbAddEventHandler(db, type, func)
     Database *db;
     int type;
     DbEventProc func;

ARGUMENTS
     db             Pointer to an open database.

     type           Specifies which event to act on.

     func           Function to call when the specified event occur.

DESCRIPTION
     During calls to DB functions that block waiting for the server to
     respond (eg. DbGetData()) events can occur to inform the application
     about the state of the request.
     Currently defined events are DbEVENT_PROGRESS.
     The DbAddEventHandler(3Db) must be called before calling the relevant
     Db request function.

SEE ALSO
     DbRemoveEventHandler(3Db)
```

```
NAME
     DbClose - disconnects a program from a database server

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     void DbClose(database)
     Database *database;

ARGUMENTS
     database       Specifies a pointer to the Database structure returned
                    from DbOpen(3Db).

DESCRIPTION
     DbClose(3Db) closes the connection between the client and the database
     server specified by database.





SEE ALSO
     DbOpen(3Db)
```

```
NAME
     DbControl - change data base handler operation

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     void DbControl(db, desc, value)
     Database *db;
     DbControlDesc *desc;
     int value;

ARGUMENTS
     db        Pointer to an open database.

     desc      Pointer to a control description structure.

     value     The desired value for the specified function.

DESCRIPTION
     Changes the data base handler operation.  Note that the new mode
     affects all requests from all connections for the specified project.

STRUCTURES
     typedef struct _DbControlDesc {
         int function;
         DbDataSpec spec;
     } DbControlDesc;

     function       The function can be one of:

          DbCONTROL_RELEASE
               The database section handling the specified
               project/instrument is expected to release any resources
               related to its data, eg. if a section keeps data files open
               it needs to close them to allow files to be removed or
               filesystems to be unmounted. The value is unused in this
               function.

          DbCONTROL_MODE
               The value can be zero or the inclusive or of
               DbMODE_REALTIME, DbMODE_BLOCK or DbMODE_SEQUENTIAL.
               Values:

               Zero - normal operation.

               DbMODE_REALTIME - if the data files are growing the internal
               knowledge of the sizes will be updated dynamically.  The
               index file will also be updated dynamically.

               DbMODE_BLOCK - if a data request is made past end of file,
               DbGetData(3Db) will block until the data becomes available.

               DbMODE_SEQUENTIAL - the request of data will not be
               controlled by time but with the DbCONTROL_SEQ function. This
               mode is useful for automatic test sequencies.

          DbCONTROL_SEQ
               Controls the operation of the sequential mode.
               Values:

               DbSEQ_FIRST - the next DbGetData(3Db) will get data from the
               beginning of the last data file.  This affects all clients
               requesting data from the same project.

               DbSEQ_LAST - the next DbGetData(3Db) will get data from the
               end of the last data file.  This affects all clients
               requesting data from the same project.

               DbSEQ_HOLD - the next and following DbGetData(3Db) will get
               its data from the same position as the previous
               DbGetData(3Db).

               DbSEQ_CONT - cancels the effect of DbSEQ_HOLD.

     spec           Data hierarchy specification.


     typedef struct _DbDataSpec {
         int project;        /* project specification (input) */
         int member;         /* project member (input) */
         int instrument;     /* project instrument (input) */
         int sensor;         /* instrument sensor (input) */
         int signal;         /* instrument signal (input) */
         int channel;        /* instrument channel (input) */
         int parameter;      /* instrument parameter (input) */
     } DbDataSpec;


RETURN VALUE
     Returns DbSUCCESS if no error occurred.  If an error occurred an error
     code is returned.
```

```
NAME
     DbDownload - download data to the data base handler

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     void DbDownload(db, desc, buffer)
     Database *db;
     DbLoadDesc *desc;
     unsigned char *buffer;

ARGUMENTS
     db        Pointer to an open database.

     desc      Pointer to a load description structure.

     buffer    Pointer to the data to be downloaded.

DESCRIPTION
     Provides a mechanism to download arbitrary data to a
     project/instrument section in the database handler.

STRUCTURES
     typedef struct _DbLoadDesc {
         DbDataSpec spec;
         int type;
         int size;
     } DbLoadDesc;

     spec           Data hierarchy specification.

     type           The data type can be one of:

          DbLOAD_TM_MAP
               Used in the Freja and Proto projects to download a telemetry
               decode map.  Each byte in the buffer must be set to one of:
               DbPROTO_CH0, DbPROTO_CH1, DbPROTO_CH2, DbPROTO_CH3,
               DbPROTO_CH4, DbPROTO_CH5 or DbPROTO_NONE.

     size           The number of bytes pointed to by buffer to download.


     typedef struct _DbDataSpec {
         int project;        /* project specification (input) */
         int member;         /* project member (input) */
         int instrument;     /* project instrument (input) */
         int sensor;         /* instrument sensor (input) */
         int signal;         /* instrument signal (input) */

         int channel;        /* instrument channel (input) */
         int parameter;      /* instrument parameter (input) */
     } DbDataSpec;
```

```
RETURN VALUE
     Returns DbSUCCESS if no error occurred.  If an error occurred an error
     code is returned.
```

```
NAME
     DbErrorString - convert an error code to an error string

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     char *DbErrorString(code)
     int code;

ARGUMENTS
     code      Specifies a database error code.

DESCRIPTION
     Converts a database error code to a null terminated error string.  The
     error string can be used in error messages to the user.


RETURN VALUE
     Returns a pointer to the error string.
```

```
NAME
     DbFreeDataObject - frees a data object

SYNOPSIS
     #include "Db.h"

     void DbFreeDataObject(ptr)
     DbDataObject *ptr;

ARGUMENTS
     ptr       Pointer to data object to be freed.

DESCRIPTION
     Frees the data returned by DbGetData(3Db).

SEE ALSO
     DbFree(3Db), DbGetData(3Db)
```

```
NAME
     DbFree - frees the data returned by other Db functions

SYNOPSIS
     #include "Db.h"

     void DbFree(ptr)
     void *ptr;

ARGUMENTS
     ptr       Pointer to data to be freed.

DESCRIPTION
     Frees the data returned by DbGetContent(3Db) and DbQuery(3Db).

SEE ALSO
     DbFreeDataObject(3Db), DbGetContent(3Db), DbQuery(3Db)
```

```
NAME
     DbGetContent - get a list of available online data

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbGetContent(db, desc, section)
     Database *db;
     DbContentDesc *desc;
     DbContentSection **section;

ARGUMENTS
     db             Pointer to an open database.

     desc           Pointer to a content description structure.

     section        Specifies a pointer that will point to a table of
                    sections on return.  Storage for the section table is
                    allocated by DbGetContent(3Db) and it is the callers
                    responsibility to free the table using DbFree(3Db) when
                    the data is no longer needed.

DESCRIPTION
     Requests a list of all data available on disk for the specified
     project and member.

STRUCTURES
     typedef struct _DbContentDesc {
         DbDataSpec spec;    /* data hierarchy specification (input) */
         int sections;       /* number of sections returned (output) */
     } DbContentDesc;

     spec           Data hierarchy specification.

     sections       Number of sections returned.


     typedef struct _DbContentSection {
         DbDataSpec spec;
         IsTimePeriod period;
         char message[32];
     } DbContentSection;


     spec           Data hierarchy description for this section.

     period         Time period of this section given as start/interval.

     message        May contain some informative message, if not it is set
                    to an empty string.

     typedef struct _DbDataSpec {
         int project;        /* project specification (input) */
         int member;         /* project member (input) */
         int instrument;     /* project instrument (input) */
```

```
        int sensor;         /* instrument sensor (input) */
        int signal;         /* instrument signal (input) */
        int channel;        /* instrument channel (input) */
        int parameter;      /* instrument parameter (input) */
    } DbDataSpec;

    typedef struct _IsTimePeriod {
        IsTime start;   /* start of time period */
        IsTime interval;     /* length of time period */
    } IsTimePeriod;

    typedef struct _IsTime { /* Isdat internal time */
        long s;             /* seconds since January 1, 1970 */
        long ns;            /* and nanoseconds */
    } IsTime;
```

```
RETURN VALUE
     Returns DbSUCCESS if no error occurred. If an error occurred no
     content is returned and an error code is returned.

ERRORS
     If an error occurs one of the following error codes is returned:

     DbBAD_PROJECT      The requested project is not available during the
                        requested interval.

     DbBAD_MEMBER       The requested member is not available during the
                        requested interval.

     DbBAD_INSTRUMENT   The requested instrument is not available during
                        the requested interval.

     DbBAD_MEMORY       Request couldn't be serviced because of memory
                        limitations.

     DbBAD_INTERNAL     Request couldn't be serviced because of some
                        internal failure.

     DbNOT_IMPLEMENTED  The requested operation is not yet implemented for
                        the given project.
```

```
NAME
     DbGetData - get specified data from the database handler

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbGetData(db, request, object)
     Database *db;
     DbDataRequest *request;
     DbDataObject **object;

ARGUMENTS
     db        Pointer to an open database.

     request   Pointer to a data request structure.

     object    Specifies a pointer that will point to the requested data
               object on return.  Storage for the object is allocated by
               this function and it is the callers responsibility to free
               the data using DbFreeDataObject(3Db) when the data object is
               no longer needed.

DESCRIPTION
     Gets data from the database handler.
     DbGetData(3Db) returns a contiguous data array, if a data gap or drop
     is present it will be filled according to the specified gap fill
     strategy.
     DbGetSegmentedData(3Db) returns a segment for each contiguous section
     of the data array, eg. if a drop is present in the data two segments
     will be returned.
     DbGetTimeTaggedData(3Db) differs from DbGetSegmentedData(3Db) in that
     a time array is returned that gives the exact time of each returned
     data sample.

STRUCTURES
     typedef struct _DbDataRequest {
         IsTime start;       /* start time of requested data */
         IsTime interval;    /* time interval of requested data */
         DbDataSpec spec;    /* data hierarchy specification */
         int units;          /* requested units */
         int reduction;      /* type of data reduction */
         int samples;        /* maximum number of data samples to return */
         int gapFill;        /* how to fill data gaps */
         int pack;           /* data pack mode */
     } DbDataRequest;

     start          Start time of the requested data.
     interval       Interval time of the requested data.

     spec           Data hierarchy specification.

     units          Defines the units of the returned data.  Possible
                    values are: DbUN_TM, DbUN_CORR and DbUN_PHYS.
```

| | |
|---|---|
| reduction | Defines the data reduction strategy used. If set to DbRED_NONE all available samples are returned. In all other cases the samples variable indicates the maximum number of samples to be returned. Possible reduction algorithms are DbRED_AVERAGE, DbRED_SKIP, DbRED_MIN, DbRED_MAX and DbRED_RESAMPLE. |
| samples | If reduction is not set to DbNONE and samples is set to a value n, at most n samples will be returned, the number of samples will reduced according to the reduction parameter. |
| gapFill | Defines how data gaps will be represented in the returned data. Gaps can be filled with IEEE NaN values (DbGAP_NAN), filled with zero values (DbGAP_ZERO) and filled with interpolated values (DbGAP_INTERPOL). Only used when pack is set to DbPACK_FILL. |
| pack | Defines the packing mode. Non-contigous data can be filled (DbPACK_FILL), ordered into contigous segments (DbPACK_SEGMENT) or each sample get it's own time tag (DbPACK_TIMETAG). |

```
typedef struct _DbDataObject {
    int rank;           /* the rank of the data type */
    int complete;       /* complete or uncomplete rank */
    int dataType;       /* type of data */
    int dimension;      /* the dimension of the data */
    int *n;             /* number of identical data types
                           in each dimension */
    int pack;           /* the data pack mode used */
    int reduction;      /* type of reduction performed */
    int gapFill;        /* how gaps were handled */
    int segments;       /* number of segments */
    DbDataSegment *seg; /* table of segments */
    int samples;        /* number of data samples */
    /* meta data */
    DbDataInfo *info;   /* info table with info for each dimension */
    int mapType;        /* the data type of each map value */
    void **map;         /* info map values for each dimension */
    IsTime **timeOffset;/* time offset values for each dimension */
    DbCoordinate coord; /* coordinate system data */
    DbDataSpec spec;    /* data hierarchy specification of returned data */
    char title[32];     /* title string to be used in plot */
    char message[64];   /* optional message */
    char version[32];   /* version string */
    unsigned int warning;/* gives caller a warning that the
                           requested data is returned but is
                           corrupted in some way */
} DbDataObject;
```

| | |
|---|---|
| rank | The rank of the data, possible values are: 0, 1, 2, 3, DbRANK_2D or DbRANK_DIAG. |
| complete | If a tensor is not a true tensor but lacks some elements it will be flagged as not complete. |

| | |
|---|---|
| dataType | The type of data returned. Available data types are DbTYPE_FLOAT, DbTYPE_COMPLEX, DbTYPE_SHORT, DbTYPE_BYTE or DbTYPE_STRING. |
| dimension | This member defines the vector dimension of the returned data. The dimension of a scalar is zero. |
| n | An array of integers giving the number of identical data types in each dimension per sample. The size of this array is dimension. If dimension = 0 it will be a null pointer. |
| pack | Defines the packing mode used. Non-contigous data can be filled (DbPACK_FILL), ordered into contigous segments (DbPACK_SEGMENT) or each sample get it's own time tag (DbPACK_TIMETAG). |
| reduction | The data reduction strategy used. If set to DbRED_NONE all available samples are returned. Possible reduction algorithms are DbRED_AVERAGE, DbRED_SKIP, DbRED_MIN, DbRED_MAX and DbRED_RESAMPLE. |
| gapFill | Defines how data gaps are represented in the data. Gaps can be filled with IEEE NaN values (DbGAP_NAN), filled with zero values (DbGAP_ZERO) and filled with interpolated values (DbGAP_INTERPOL). |
| segments | Number of data segments in the data object. |
| seg | Pointer to a table of segment descriptors. Each segment is sequence of contiguous data samples. |
| samples | The number of samples in the data object. |
| info | Info table with info for each dimension. The size of this array is (dimension + 1). |
| mapType | The data type of the map values. Possible values are DbTYPE_FLOAT and DbTYPE_STRING. |
| map | Info map values for each dimension mapping each index in that dimension to a physical value. The size of each array is n[0], n[1], ..., n[dimension - 1]. It is a NULL pointer if dimension = 0. |
| timeOffset | The time offset for each data point corresponding to the dimension and index value. The size of each array is n[0], n[1], ..., n[dimension - 1]. It is a NULL pointer if dimension = 0. |
| coord | Coordinate system data. |
| spec | Data hierarchy specification also called the logical instrument.. |
| title | Title string that can be used to label plots. |

```
message        Optional message.

version        The combined versions of all modules involved in
               producing the data. It will be a hierarchial
               versioning, eg. "2.0.3.5.43" which states that the
               ISDAT version is 2.0, the instrument module version is
               3.5 and the calibration version used was 43.

warning        On return this member is set to indicate in which way
               the returned data is corrupted. Each reason is coded as
               a bit mask and one call can result in several warning
               conditions to be set. The defined warnings are: the
               experiment mode matches the requested criteria only
               part of the requested interval (DbWARN_PART), a data
               drop occurred in the interval (DbWARN_DROP), a gap is
               present in the interval (DbWARN_GAP), some part of the
               interval is before the beginning of the file
               (DbWARN_BOF) and an end of file occurred somewhere in
               the requested interval (DbWARN_EOF).  The gaps will be
               filled according to the gap fill strategy defined.

               A drop is flagged when data is missing because of some
               error. A gap is flagged when the data set is designed
               with gaps in between data.

typedef struct _DbDataSegment {
    IsTime start;       /* start time of this segments data */
    IsTime interval;    /* time interval of this segments data */
    int samples;        /* number of data samples in the segment */
    void *data;         /* pointer to an array of the actual data */
    IsTime *time;       /* time line, one timetag per sample */
} DbDataSegment;


start          Start time of the data.

interval       Interval time of the data.

samples        The number of samples in this segment.

data           The data array. The pointer has to be cast into the
               appropriate data type depending on the value of rank,
               complete and dataType.

time           Time line with one timetag per sample.  Only valid if
               pack = DbPACK_TIMETAG.
typedef struct _DbDataInfo {
    int units;          /* physical units of returned data */
    int quantity;       /* quantity descriptor */
    int scaleType;      /* type of scale */
    float scaleMin;     /* min value of data */
    float scaleMax;     /* max value of data */
    float samplingFreq; /* sampling frequency used */
    float filterFreq;   /* filter frequency used */
    char unitString[32];/* physical units of returned data */
    char quantityString[32];/* quantity string */
    char conversion[80];/* SI conversion string */
} DbDataInfo;
```

| | |
|---|---|
| units | Defines the units of the data. Possible values are: DbUN_TM, DbUN_CORR, DbUN_V_PER_M, DbUN_MV_PER_M, DbUN_PROCENT, DbUN_MV_PER_M_SQR_PER_HZ, DbUN_MICRO_AMP, DbUN_NANO_TESLA and DbUN_DECIBELL. |
| quantity | Description of quantity associated with the data. Possible values are DbQTY_FREQUENCY, DbQTY_POWER, DbQTY_COUNTS, DbQTY_ENERGY and DbQTY_ANGLE. |
| scaleType | Type of scale, DbSCALE_LIN, DbSCALE_LOG or DbSCALE_IRREGULAR. |
| scaleMin | A value less or equal to the minimum data point. To be used as a hint for plotting. |
| scaleMax | A value greater or equal to the maximum data point. To be used as a hint for plotting. |
| samplingFreq | The sample frequency used by the experiment. |
| filterFreq | The filter frequency used by the experiment. |
| unitString | Unit string to be used in plots. |
| quantityString | Quantity string to be used in plots. |
| conversion | SI conversion string. |

```
typedef struct _DbCoordinate {
    int system;         /* coordinate system */
    DbDataR2 rot;       /* rotation matrix */
} DbCoordinate;
```

| | |
|---|---|
| system | Coordinate system of returned data, DbCOORD_SENSOR, DbCOORD_PLATFORM, DbCOORD_DESPUN or DbCOORD_GSE. |
| rot | Rotation matrix with respect to DbCOORD_PLATFORM. |

```
typedef struct _DbDataSpec {
    int project;        /* project specification (input) */
    int member;         /* project member (input) */
    int instrument;     /* project instrument (input) */
    int sensor;         /* instrument sensor (input) */
    int signal;         /* instrument signal (input) */
    int channel;        /* instrument channel (input/output) */
    int parameter;      /* instrument parameter (input/output) */
} DbDataSpec;
```

| | |
|---|---|
| project | Project specification. Can be one of DbVIKING, DbFREJA, |

| | |
|---|---|
| | DbCLUSTER, DbCSDS_SP, DbCSDS_PP, DbPROTO and DbEISCAT. |
| member | Project member. This field is only used for the Cluster and Eiscat projects. Valid values are C1, C2, C3 and C4 for Cluster and DbEIS_TROMSO, DbEIS_KIRUNA and DbEIS_SODANKYLA for Eiscat. |
| instrument | Project instrument. Viking instruments are DbVIK_V2, DbVIK_V3, DbVIK_V4L and DbVIK_V4H. Cluster instruments are DbCLU_EFW and DbCLU_STAFF. Eiscat instruments are DbEIS_VHF and DbEIS_UHF. |
| sensor | Instrument sensor. Viking V2 sensors are DbVIK2_BX, DbVIK2_BY and DbVIK2_BZ. Viking V3 sensors are DbVIK3_PISP1 and DbVIK3_PISP2. Viking V4L sensors are DbVIK4_EX, DbVIK4_EY, DbVIK4_EZ, DbVIK4_DBX, DbVIK4_N1 and DbVIK4_N2. Proto sensors are DbPROTO_CH0, DbPROTO_CH1, DbPROTO_CH2, DbPROTO_CH3, DbPROTO_CH4 and DbPROTO_CH5. Eiscat sensors are tbd. |
| signal | Instrument signal. Viking V4L signals are DbVIK4_WF, DbVIK4_DFT and DbVIK4_FB. Viking V4H signals are DbVIK4_FB. Eiscat signals are tbd. |
| channel | Instrument channel. Viking V4L filter bank channels are DbVIK4_500HZ, DbVIK4_1KHZ and DbVIK4_2KHZ. Viking V4H filter bank channels are DbVIK4_4KHZ, DbVIK4_8KHZ, DbVIK4_16KHZ, DbVIK4_32KHZ, DbVIK4_64KHZ, DbVIK4_128KHZ, DbVIK4_256KHZ or DbVIK4_512KHZ. |
| parameter | Instrument parameter. |

```
typedef struct _IsTime { /* define Isdat time (IsTime) */
    long s;            /* seconds since January 1, 1970 */
    long ns;           /* and nanoseconds */
} IsTime;
```

RETURN VALUE

Returns DbSUCCESS if no error occurred. If an error occurred no data is returned and an error code is returned.

ERRORS

If an error occurs one of the following error codes is returned:

| | |
|---|---|
| DbBAD_TIME | Requested time is not found on the disc. |
| DbBAD_PROJECT | The requested project is not available during the requested interval. |
| DbBAD_MEMBER | The requested member is not available during the requested interval. |
| DbBAD_INSTRUMENT | The requested instrument is not available during the requested interval. |
| DbBAD_SENSOR | The requested sensor is not available during the |

```
                        requested interval.

     DbBAD_SIGNAL       The requested signal is not available during the
                        requested interval.

     DbBAD_CHANNEL      The requested channel is not available during the
                        requested interval.
     DbBAD_PARAMETER    The requested parameter is not available during
                        the requested interval.

     DbBAD_UNITS        The requested units is not valid.

     DbBAD_REDUCTION    The requested reduction is not valid.

     DbBAD_GAPFILL      The requested gapfill is not valid.

     DbBAD_ALLOC        Request couldn't be serviced because of memory
                        limitations.

     DbBAD_INTERNAL     Request couldn't be serviced because of some
                        internal failure.

     DbNOT_IMPLEMENTED  The requested operation is not yet implemented for
                        the given project.
SEE ALSO
     DbFreeDataObject(3Db)
```

```
NAME
     DbGetInfo - get information about the specified data hierarchy object

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbGetInfo(db, desc, data)
     Database *db;
     DbInfoDesc *desc;
     DbInfoData **data;

ARGUMENTS
     db        Pointer to an open database.

     desc      Pointer to a info description structure.

     data      Specifies a pointer that will point to the requested data on
               return.  Storage for the data is allocated by this function
               and it is the callers responsibility to free the data using
               DbFree(3Db) when the data is no longer needed.

DESCRIPTION
     Get type and coordinate information about the specified data hierarchy
     object, eg. a sensor.

STRUCTURES
     typedef struct _DbInfoDesc {
         DbDataSpec spec;
     } DbInfoDesc;

     spec           Data hierarchy specification. Unused fields must be set
                    to DbUNUSED.

     typedef struct _DbInfoData {
         int category;
         DbInfoCoord location;
         DbInfoCoord direction;
     } DbInfoData;


     category       The category of the object.

     location       The location of the object in spacecraft coordinates.

     direction      The pointing direction of the object in spacecraft
                    coordinates.

     typedef struct _DbInfoCoord {
         int valid;
         float x;
         float y;
         float z;
     } DbInfoCoord;
```

```
valid          Set to one if the coordinates are valid.

x              X coordinate.

y              Y coordinate.

z              Z coordinate.

typedef struct _DbDataSpec {
    int project;
    int member;
    int instrument;
    int sensor;
    int signal;
    int channel;
    int parameter;
} DbDataSpec;
```

RETURN VALUE

Returns DbSUCCESS if no error occurred. If an error occurred no data is returned and an error code is returned.

```
NAME
     DbName2Spec - convert a string specification to a data specification

SYNOPSIS
     #include "Db.h"

     int DbName2Spec(db, name, spec)
     Database *db;
     DbSpecName *name;
     DbDataSpec *spec;

ARGUMENTS
     db        Pointer to an open database.

     name      Pointer to a structure of data specification name strings.

     spec      Pointer to a data specification structure.

DESCRIPTION
     Converts from a name specification pointed to by name to a data
     specification pointed to by spec;

STRUCTURES
     typedef struct _DbDataSpec {
         int project;        /* project specification (input) */
         int member;         /* project member (input) */
         int instrument;     /* project instrument (input) */
         int sensor;         /* instrument sensor (input) */
         int signal;         /* instrument signal (input) */
         int channel;        /* instrument channel (input) */
         int parameter;      /* instrument parameter (input) */ }
     DbDataSpec;

     typedef struct _DbSpecName {
         char project[16];   /* project name (output) */
         char member[16];    /* project member name (output) */
         char instrument[16];/* project instrument name (output) */
         char sensor[16];    /* instrument sensor name (output) */
         char signal[16];    /* instrument signal name (output) */
         char channel[16];   /* instrument channel name (output) */
         char parameter[16]; /* instrument parameter name (output) */ }
     DbSpecName;


RETURN VALUE
     Returns DbSUCCESS on successful completion.
```

```
NAME
     DbName - report the database name when connection to a database fails

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     char *DbName(string)
     char *string;

ARGUMENTS
     string    Specifies the character string.

DESCRIPTION
     DbName(3Db) is normally used to report the name of the database the
     program attempted to open with DbOpen(3Db). If a NULL string is
     specified, DbName(3Db) looks in the environment for DATABASE and
     returns the database name that the user was requesting. Otherwise it
     returns its own argument.


RETURN VALUE
     Returns a pointer to the reported name.




SEE ALSO
     DbOpen(3Db)
```

```
NAME
     DbOpen - connect a program to a database server

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     Database *DbOpen(databaseName, argc, argv)
     char *databaseName;
     int argc;
     char **argv;

ARGUMENTS
     databaseName   Specifies the database name, which determines the
                    database and communications domain to be used. May be a
                    NULL pointer.

     argc           Number of arguments in argc.

     argv           Argument list from main() to enable DbOpen(3Db) to
                    parse command line arguments.

DESCRIPTION
     The DbOpen(3Db) routine connects the client to a database server
     through TCP, UNIX or DECnet streams.

     If databaseName is NULL, the value defaults to the contents of the
     ISDAT_DATABASE environment variable. The databaseName or
     ISDAT_DATABASE environment variable is a string that has the format
     hostname:database[.baseport].  For example, irfu:2 would specify
     database server 2 on the machine irfu.

     hostname       Specifies the name of the host machine on which the
                    database server runs.  You follow the hostname with
                    either a single colon (:) or a double colon (::), which
                    determines the communications domain to use.  Any or
                    all of the communications protocols can be used
                    simultaneously on a server built to support them.

                    If hostname is a host machine and a single colon (:)
                    separates the hostname and database number, TCP streams
                    is used for the connection.

                    If hostname is "unix" and a single colon (:) separates
                    it from the database number, UNIX domain IPC streams is
                    used for the connection.

                    If hostname is a host machine and a double colon (::)
                    separates the hostname and database number, DECnet
                    streams is used for the connection.

     database       Specifies the number of the database server on its host
                    machine.  A single CPU can have more than one database;
                    the databases are numbered starting from 0.
```

```
    baseport       Optional argument to change the TCP/IP base port
                   number.  For example, irfu:2.20000 would specify
                   database server 2 on the machine irfu using the base
                   port number 20000, the resulting port number will be
                   20002.
                   If baseport is not defined or set to zero the default
                   baseport 14734 will be used.


RETURN VALUE
    Returns a pointer to a Database structure if successful. If an error
    occurs, it returns NULL.


SEE ALSO
    DbClose(3Db)
```

```
NAME
     DbOverview - get an overview of available online data matching
     specification and event

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbOverview(db, desc, section)
     Database *db;
     DbOverviewDesc *desc;
     DbOverviewSection **section;

ARGUMENTS
     db             Pointer to an open database.

     desc           Pointer to a overview description structure.

     section        Specifies a pointer that will point to a table of
                    sections on return.  Storage for the section table is
                    allocated by DbOverview(3Db) and it is the callers
                    responsibility to free the table using DbFree(3Db) when
                    the data is no longer needed.

DESCRIPTION
     A start/interval is given together with a complete data hierarchy
     specification and an event, a detailed description will be returned
     for each matching data set.  One section is created for each data set
     that matches spec.

STRUCTURES
     typedef struct _DbOverviewDesc {
         IsTime start;       /* when to start overview (input) */
         IsTime interval;    /* time interval of overview (input) */
         DbDataSpec spec;    /* data hierarchy specification (input) */
         unsigned int event; /* event specification */
         int sections;       /* number of sections returned (output) */
     } DbOverviewDesc;

     start          Start time of the requested overview.

     interval       Time interval of the requested overview.

     spec           Data hierarchy specification.  The value DbUNDEF can be
                    used as wildcard to match anything.

     event          Set to zero if no events are to be reported.
                    Events can be one of DbEVENT_SWEEP, DbEVENT_CALIBRATION
                    or DbEVENT_SOUNDER.
     sections       Number of sections returned.


     typedef struct _DbOverviewSection {
         DbDataSpec spec;
         int items;
```

```
    IsTimePeriod *period;
    char message[32];
} DbOverviewSection;
```

| | |
|---|---|
| spec | Data hierarchy description for this section. |
| items | Number of periods in the array pointed to by period. |
| period | Points to an array of period (start/interval) values. |
| message | May contain some informative message, if not it is set to an empty string. |

```
typedef struct _DbDataSpec {
    int project;        /* project specification (input) */
    int member;         /* project member (input) */
    int instrument;     /* project instrument (input) */
    int sensor;         /* instrument sensor (input) */
    int signal;         /* instrument signal (input) */
    int channel;        /* instrument channel (input) */
    int parameter;      /* instrument parameter (input) */
} DbDataSpec;

typedef struct _IsTimePeriod {
    IsTime start;   /* start of time period */
    IsTime interval;     /* length of time period */
} IsTimePeriod;

typedef struct _IsTime { /* Isdat internal time */
    long s;             /* seconds since January 1, 1970 */
    long ns;            /* and nanoseconds */
} IsTime;
```

RETURN VALUE

Returns DbSUCCESS if no error occurred. If an error occurred no sections are is returned and an error code is returned.

ERRORS

If an error occurs one of the following error codes is returned:

| | |
|---|---|
| DbBAD_TIME | Requested time is not found. |
| DbBAD_PROJECT | The requested project is not available during the requested interval. |
| DbBAD_MEMBER | The requested member is not available during the requested interval. |
| DbBAD_INSTRUMENT | The requested instrument is not available during the requested interval. |
| DbBAD_SENSOR | The requested sensor is not available during the requested interval. |
| DbBAD_SIGNAL | The requested signal is not available during the |

| | |
|---|---|
| | requested interval. |
| DbBAD_CHANNEL | The requested channel is not available during the requested interval. |
| DbBAD_PARAMETER | The requested parameter is not available during the requested interval. |
| DbBAD_MEMORY | Request couldn't be serviced because of memory limitations. |
| DbBAD_INTERNAL | Request couldn't be serviced because of some internal failure. |
| DbNOT_IMPLEMENTED | The requested operation is not yet implemented for the given project. |

```
NAME
     DbPrepareData - prepare a data set before use

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbPrepareData(db, desc)
     Database *db;
     DbPrepareDesc *desc;

ARGUMENTS
     db       Pointer to an open database.

     desc     Pointer to a prepare data description structure.

DESCRIPTION
     Prepares data for the given time span. Some implementations require
     that DbPrepareData(3Db) gets called before any call to DbGetData(3Db).

STRUCTURES
     typedef struct _DbPrepareDesc {
         IsTime start;       /* start time of requested data (input/output) */
         IsTime interval;    /* time interval of requested data (input/output) */
         DbDataSpec spec;    /* data hierarchy specification (input) */
     } DbPrepareDesc;

     start          Start time of the data to be preparred. The value may
                    be changed by the call.

     interval       Interval time of the data to be preparred. The value
                    may be changed by the call.

     spec           Data hierarchy specification.

     typedef struct _DbDataSpec {
         int project;        /* project specification (input) */
         int member;         /* project member (input) */
         int instrument;     /* project instrument (input) */
         int sensor;         /* instrument sensor (input) */
         int signal;         /* instrument signal (input) */
         int channel;        /* instrument channel (input) */
         int parameter;      /* instrument parameter (input) */
     } DbDataSpec;


     project        Project specification.

     member         Project member. This field is only used for the Cluster
                    and Eiscat projects.

     instrument     Project instrument.

     sensor         Instrument sensor.
```

| | |
|---|---|
| signal | Instrument signal. |
| channel | Instrument channel. |
| parameter | Instrument parameter. |

```
typedef struct _IsTime { /* define Isdat time (IsTime) */
    long s;            /* seconds since January 1, 1970 */
    long ns;           /* and nanoseconds */
} IsTime;
```

RETURN VALUE

Returns DbSUCCESS if no error occurred. If an error occurred an error code is returned.

ERRORS

If an error occurs one of the following error codes is returned:

| | |
|---|---|
| DbBAD_TIME | Requested time is not found on the disc. |
| DbBAD_PROJECT | The requested project is not available during the requested interval. |
| DbBAD_MEMBER | The requested member is not available during the requested interval. |
| DbBAD_INSTRUMENT | The requested instrument is not available during the requested interval. |
| DbBAD_SENSOR | The requested sensor is not available during the requested interval. |
| DbBAD_SIGNAL | The requested signal is not available during the requested interval. |
| DbBAD_CHANNEL | The requested channel is not available during the requested interval. |
| DbBAD_PARAMETER | The requested parameter is not available during the requested interval. |
| DbBAD_UNITS | The requested units is not valid. |
| DbBAD_REDUCTION | The requested reduction is not valid. |
| DbBAD_GAPFILL | The requested gapfill is not valid. |
| DbBAD_ALLOC | Request couldn't be serviced because of memory limitations. |
| DbBAD_INTERNAL | Request couldn't be serviced because of some internal failure. |
| DbNOT_IMPLEMENTED | The requested operation is not yet implemented for the given project. |

NAME

    DbQuantityString - convert a quantity value to a printable string

SYNOPSIS

```
#include "Db.h"

char *DbQuantityString(quantity)
int quantity;
```

ARGUMENTS

| | |
|---|---|
| quantity | Quantity value. |

DESCRIPTION

    Converts the specified quantity value to its corresponding name string, eg. DbQTY_ENERGY will return the string "energy".

RETURN VALUE

    Returns the quantity name string. If an invalid quantity value is specified, the string "undefined quantity" is returned.

```
NAME
     DbQuery - get database data hierarchy description

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbQuery(db, desc, qdata)
     Database *db;
     DbQueryDesc *desc;
     DbQueryData **qdata;

ARGUMENTS
     db        Pointer to an open database.

     desc      Pointer to a query description.

     qdata     Specifies a pointer that will point to an DbQueryData array
               on return.  The last element in the array will have value
               set to -1 and name set to NULL.  It is the callers
               responsibility to free the array using DbFree(3Db) when the
               data is no longer needed.  Pointer to a query description.

DESCRIPTION
     This call enables the user to query the database for all available
     data description choices at a specified level.  This can be used to
     write programs that can operate on different projects / instruments
     without any knowledge about them.

STRUCTURES
     typedef struct _DbQueryDesc {
         int mode;       /* Must be set to DbALL. Currently not used. */
         int level;      /* One of DbLEVEL_PROJECT, DbLEVEL_MEMBER,
                            DbLEVEL_INSTRUMENT, DbLEVEL_SENSOR,
                            DbLEVEL_SIGNAL, DbLEVEL_CHANNEL
                            or DbLEVEL_PARAMETER */
         IsTime time;    /* Currently not used. */
         DbDataSpec spec;/* data hierarchy specification */
     } DbQueryDesc;

     typedef struct _DbDataSpec {
         int project;       /* Project specification, only needed if level
                               is set to DbMEMBER or higher */
         int member;        /* Member specification, only needed if level
                               is set to DbINSTRUMENT or higher */
         int instrument;    /* Instrument specification, only needed if level
                               is set to DbSENSOR or higher */
         int sensor;        /* Sensor specification, only needed if level
                               is set to DbSIGNAL or higher */
         int signal;        /* Signal specification, only needed if level
                               is set to DbCHANNEL */
         int channel;       /* Signal specification, only needed if level
                               is set to DbPARAMETER */
         int parameter;     /* not used */
     } DbDataSpec;
```

```
typedef struct _DbQueryData {
    int value;      /* Value to be used at the specified level to request
                       data from the database */
    int groupId;    /* entries with the same number within the array
                       group together (eg. magnetometer x, y ,z),
                 if groupId is zero the entry doesn't group together */
    char *name;     /* Symbolic name for the value. Can be used to
                       label menus and plots */
} DbQueryData;
```

RETURN VALUE

Returns DbSUCCESS if no error occurred. If an error occurred no content is returned and an error code is returned.

ERRORS

If an error occurs one of the following error codes is returned:

| | |
|---|---|
| DbBAD_ALLOC | Request couldn't be serviced because of memory limitations. |

SEE ALSO

DbFree(3Db), DbGetData(3Db)

```
NAME
     DbRemoveEventHandler - removes event handler function

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     Database *DbRemoveEventHandler(db, type)
     Database *db;
     int type;

ARGUMENTS
     db             Pointer to an open database.

     type           Specifies which event to remove.

DESCRIPTION
     Removes the current event handler for the event specified by type.
     Currently defined events are DbEVENT_PROGRESS.

SEE ALSO
     DbAddEventHandler(3Db)
```

```
NAME
     DbSpec2Name - convert a data specification to printable strings

SYNOPSIS
     #include "Db.h"

     int DbSpec2Name(db, spec, name)
     Database *db;
     DbDataSpec *spec;
     DbSpecName *name;

ARGUMENTS
     db        Pointer to an open database.

     spec      Pointer to a data specification structure.

     name      Pointer to a structure of data specification name strings.

DESCRIPTION
     Reads a data specification and converts it to printable strings.  Some
     strings may be empty if that specification level is unused.

STRUCTURES
     typedef struct _DbDataSpec {
         int project;        /* project specification (input) */
         int member;         /* project member (input) */
         int instrument;     /* project instrument (input) */
         int sensor;         /* instrument sensor (input) */
         int signal;         /* instrument signal (input) */
         int channel;        /* instrument channel (input) */
         int parameter;      /* instrument parameter (input) */ }
     DbDataSpec;

     typedef struct _DbSpecName {
         char project[16];   /* project name (output) */
         char member[16];    /* project member name (output) */
         char instrument[16];/* project instrument name (output) */
         char sensor[16];    /* instrument sensor name (output) */
         char signal[16];    /* instrument signal name (output) */
         char channel[16];   /* instrument channel name (output) */
         char parameter[16]; /* instrument parameter name (output) */ }
     DbSpecName;


RETURN VALUE
     Returns DbSUCCESS on successful completion.
```

```
NAME
     DbUnitString - convert a unit value to a printable string

SYNOPSIS
     #include "Db.h"

     char *DbUnitString(unit)
     int unit;

ARGUMENTS
     unit      Unit value.

DESCRIPTION
     Converts the specified unit value to its corresponding name string,
     eg. DbUN_DECIBELL will return the string "dB".


RETURN VALUE
     Returns the unit name string. If an invalid unit value is specified,
     the string "undefined unit" is returned.
```

```
NAME
     DbUpload - upload data from the data base handler

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     void DbUpload(db, desc, buffer)
     Database *db;
     DbLoadDesc *desc;
     unsigned char **buffer;

ARGUMENTS
     db       Pointer to an open database.

     desc     Pointer to a load description structure.

     buffer   Specifies a pointer that will point to the requested data on
              return.  Storage for the data is allocated by DbUpload(3Db)
              and it is the callers responsibility to free the data using
              DbFree(3Db) when the data is no longer needed.

DESCRIPTION
     Provides a mechanism to upload arbitrary data from a
     project/instrument section in the database handler.

STRUCTURES
     typedef struct _DbLoadDesc {
         DbDataSpec spec;
         int type;
         int size;
     } DbLoadDesc;

     spec           Data hierarchy specification.

     type           The data type can be one of:

          DbLOAD_TM_MAP
              Used in the Freja and Proto projects to upload the current
              telemetry decode map.  Each byte in the buffer will be set
              to one of: DbPROTO_CH0, DbPROTO_CH1, DbPROTO_CH2,
              DbPROTO_CH3, DbPROTO_CH4, DbPROTO_CH5 or DbPROTO_NONE.

     size           The size of the returned data in bytes.


     typedef struct _DbDataSpec {
         int project;       /* project specification (input) */
         int member;        /* project member (input) */
         int instrument;    /* project instrument (input) */
         int sensor;        /* instrument sensor (input) */
         int signal;        /* instrument signal (input) */
         int channel;       /* instrument channel (input) */
         int parameter;     /* instrument parameter (input) */
     } DbDataSpec;
```

```
RETURN VALUE
     Returns DbSUCCESS if no error occurred.  If an error occurred an error
     code is returned.
```

```
NAME
     DbWarningString - convert a warning mask to a string

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     char *DbWarningString(mask)
     int mask;

ARGUMENTS
     mask      Specifies a database warning mask.

DESCRIPTION
     Converts a database warning mask to a null terminated string of
     concatenated warning messages. Each message is separated by a comma.
     The warning string can be used in warning messages to the user.


RETURN VALUE
     Returns a pointer to the warning string.
```

# B Islib manual pages

```
NAME
     IsAddCallback - add callback procedure

SYNOPSIS
     #include "Is.h"

     void IsAddCallback(reason, callback, closure)
     int reason;
     IsCallbackProc callback;
     IsPointer closure;

ARGUMENTS
     reason    Specifies the reason for calling the callback procedure.

     callback  Specifies the callback procedure.

     closure   Specifies the argument that is to be passed to the specified
               procedure when it is invoked. Use NULL if not used.

DESCRIPTION
     Adds the specified callback procedure.

NOTES
     Defined values for reason are:

               IsCR_TM_INFO
               IsCR_NEW_CLIENT
```

```
                IsCR_CLIENTS_DONE
                IsCR_SELECTIVE_REDRAW
                IsCR_CHANGE_TIME


SEE ALSO
     IsCallCallbacks(3Is)
```

```
NAME
     IsCallCallbacks - process callbacks

SYNOPSIS
     #include "Is.h"

     void IsCallCallbacks(reason, call_data)
     int reason;
     IsPointer call_data;

ARGUMENTS
     reason    Specifies the reason for calling the callback procedure.

     call_data Specifies a pointer to data specific to each reason that is
               passed to the callback procedures.

DESCRIPTION
     Calls each procedure that is registered in the callback list.

NOTES
     Defined values for reason are:

               IsCR_TM_INFO
               IsCR_NEW_CLIENT
               IsCR_CLIENTS_DONE
               IsCR_SELECTIVE_REDRAW
               IsCR_CHANGE_TIME


SEE ALSO
     IsAddCallback(3Is)
```

```
NAME
     IsCallPipe - calls a filter pipe

SYNOPSIS
     #include "Is.h"

     void IsCallPipe(widget, name, desc, data)
     Widget widget;
     char *name;
     IsPipeDesc *desc;
     float **data;

ARGUMENTS
     widget

     name

     desc

     data

DESCRIPTION
     Calls a filter pipe on the drawing area specified by widget and name
     specified by name.

STRUCTURES
     typedef struct _IsPipeDesc {
         int type;           /* type of data */
         int dimension;      /* data vector dimension (input/output) */
         int samples[5];     /* number of data samples (input/output) */
     } IsPipeDesc;

     type           The data type, eg. IsPIPE_FLOAT or IsPIPE_ASCII.

     dimension      The vector dimension of the data.

     samples        The number of samples in each dimension.

NOTES
     This is a client only function.

SEE ALSO
     IsRegisterPipe(3Is)
```

```
NAME
     IsChangeTime - change time manager time

SYNOPSIS
     #include "Is.h"

     void IsChangeTime(msg)
     IsTimeMessage *msg;

DESCRIPTION
     Tells the time manager to change time and interval.

NOTES
     This is a client only function.
```

```
NAME
     IsClientExec - execute a client

SYNOPSIS
     #include "Is.h"

     void IsClientExec(clientName)
     char *clientName;

ARGUMENTS
     clientName     Name of client to execute.

DESCRIPTION
     Executes the client clientName as a child to the current process.  The
     directory used to hold clients is $HOME/isdat/bin/clients.

     The client is executed as:

               clientName parentArgs -managerWindow window

     ParentArgs are all arguments that were passed to the current process
     (the manager). Window is the window id in the manager where the client
     sends all messages.

NOTES
     This is an time manager function.

SEE ALSO
     IsExec(3Is), IsManager(3Is)
```

```
NAME
     IsClientNotify - notify and send information to the client(s)

SYNOPSIS
     #include "Is.h"

     void IsClientNotify(clientId, tmInfo)
     IsClientId *clientId;
     IsTmInfo *tmInfo;

ARGUMENTS
     clientId  Client identifier. Managers will get client identifiers when
               the IsCR_NEW_CLIENT callback procedure is called.
               If clientId is set to IsNOTIFY_ALL, all clients known to
               this manager will be notified.

     tmInfo    Informs the client what to do.

DESCRIPTION
     This function is used by a manager to notify the client(s) to do new
     analysis using the passed information.

STRUCTURES
     typedef struct _IsTmInfo {
          int project;     /* which project, eg. DbViking */
          int member;      /* which project member */
          IsTime start;    /* requested analysis start time */
          IsTime interval; /* requested analysis time interval */
          IsTime contEnd;  /* stop time of continuous mode, when continuous
                              mode is disabled it is set to start + interval */
     } IsTmInfo;

NOTES
     The project and member fields received by the client will never
     change, they are always set the values used when the client was
     started.

     This is a time manager function.
```

```
NAME
     IsClientPath - client directory path name

SYNOPSIS
     #include "Is.h"

     char *IsClientPath()

DESCRIPTION
     Returns the path name of the directory where the clients reside.

RETURN VALUE
     Returns a pointer to the path name.

NOTES
     This is an time manager function.
```

NAME

IsCreateSystemMenu - create a system drawing menu

SYNOPSIS

```
#include "Is.h"

void IsCreateSystemMenu(wid)
Widget wid;
```

ARGUMENTS

| | |
|---|---|
| wid | Specifies the widget. |

DESCRIPTION

Create a system menu and attach it to the specified wid.

NOTES

This is a client only function.

NAME
    IsExec - execute a program

SYNOPSIS
```
#include "Is.h"

void IsExec(name)
char *name;
```

ARGUMENTS

| | |
|---|---|
| name | Name of program to execute. |

DESCRIPTION
    Executes the program name as a child to the current process. The directory used to hold programs is $HOME/isdat/bin.

NOTES
    This is an time manager function.

SEE ALSO
    IsClientExec(3Is)

```
NAME
     IsFilter - act as a filter

SYNOPSIS
     #include "Is.h"

     void IsFilter()

DESCRIPTION
     Tells the isdat interface that this program is a filter.
     This function must be the first isdat interface function to be called
     in a filter program.

SEE ALSO
     IsInitialize(3Is)
```

```
NAME
     IsGetTmInfo - get the latest time manager information

SYNOPSIS
     #include "Is.h"

     IsTmInfo *IsGetTmInfo()

DESCRIPTION
     Gets the latest information sent by the time manager to the client.

STRUCTURES
     typedef struct _IsTmInfo {
          int project;     /* which project, eg. IsViking */
          int member;      /* which project member */
          IsTime start;    /* requested analysis start time */
          IsTime interval; /* requested analysis time interval */
          IsTime contEnd;  /* stop time of continuous mode, when continuous
                              mode is disabled it is set to start + interval */
     } IsTmInfo;

RETURN VALUE
     Returns a pointer to a valid IsTmInfo structure.  If no time manager
     information has been received yet, NULL is returned.

NOTES
     This is a client only function.

SEE ALSO
     IsClientNotify(3Is)
```

```
NAME
     IsInitialize - initialize the user interface

SYNOPSIS
     #include "Is.h"

     void IsInitialize(argc, argv, dpy)
     int argc;
     char **argv;
     Display *dpy;

ARGUMENTS
     argc      Number of arguments. Same as argc in main().

     argv      Pointer to a table of argument strings.  Same as argv in
               main().

     dpy       Pointer to an open X-window display.  If the Ui library is
               used it is returned by UiInitialize(3Ui).

DESCRIPTION
     In a client, IsInitialize(3Is) sets up the communication to talk to
     the time manager.

     In an time manager, IsInitialize(3Is) sets up the communication to
     talk to the clients.

SEE ALSO
     IsManager(3Is)
```

```
NAME
     IsMainLoop - get and process events

SYNOPSIS
     #include "Is.h"

     void IsMainLoop()

DESCRIPTION
     Handle events and process them.  IsMainLoop will never return and is
     therefore normally the last function in the program.
```

```
NAME
     IsManager - act as an time manager

SYNOPSIS
     #include "Is.h"

     void IsManager()

DESCRIPTION
     Tells the isdat interface that this program is an time manager.
     This function must be the first isdat interface function to be called
     in a time manager.

SEE ALSO
     IsInitialize(3Is)
```

```
NAME
     IsPipeRead - filter function to read data

SYNOPSIS
     #include "Is.h"

     void IsPipeRead(desc, buffer)
     IsPipeDesc *desc;
     float **buffer;

ARGUMENTS
     desc

     samples

DESCRIPTION
     Function used in a filter to read data to process.

NOTES
     This is a filter only function.

SEE ALSO
     IsPipeRead(3Is)
```

```
NAME
     IsPipeWrite - filter function to write data

SYNOPSIS
     #include "Is.h"

     void IsPipeWrite(desc, buffer)
     IsPipeDesc desc;
     float *buffer;

ARGUMENTS
     desc

     buffer

DESCRIPTION
     Function used in a filter to write back processed data.

NOTES
     This is a filter only function.

SEE ALSO
     IsPipeWrite(3Is)
```

```
NAME
     IsRedrawMe - redraw request

SYNOPSIS
     #include "Is.h"

     void IsRedrawMe()

DESCRIPTION
     Tells the time manager to repeat the last information it sent. This
     will generate an IsCR_TM_INFO callback and cause a redraw.

NOTES
     This is a client only function.

SEE ALSO
     IsClientNotify(3Is)
```

```
NAME
     IsRegisterPipe - register a filter pipe

SYNOPSIS
     #include "Is.h"

     void IsRegisterPipe(widget, name)
     Widget widget;
     char *name;

ARGUMENTS
     widget

     name

DESCRIPTION
     Registers a filter pipe on the drawing area specified by widget and
     gives it the name specified by name.

NOTES
     This is a client only function.

SEE ALSO
     IsCallPipe(3Is)
```

```
NAME
     IsSetTmInfo - set the time manager information

SYNOPSIS
     #include "Is.h"

     void IsSetTmInfo(info)
     IsTmInfo *info

ARGUMENTS
     *info

DESCRIPTION
     Sets information that can be read by the client before it enters the
     main loop.

STRUCTURES
     typedef struct _IsTmInfo {
          int project;     /* which project, eg. DbViking */
          int member;      /* which project member */
          IsTime start;    /* requested analysis start time */
          IsTime interval; /* requested analysis time interval */
          IsTime contEnd;  /* stop time of continuous mode, when continuous
                        mode is disabled set it to start + interval */
     } IsTmInfo;

NOTES
     This is a manager only function.

SEE ALSO
     IsGetTmInfo(3Is), IsClientNotify(3Is)
```

## C Isutillib manual pages

```
NAME
     IsAddTimeDouble - adds a float to a time value

SYNOPSIS
     #include "Isutil.h"

     void IsAddTimeDouble(a, seconds)
     IsTime *a;
     double seconds;

ARGUMENTS
     a         Pointer to an IsTime structure.

     seconds   Number of seconds to add.

DESCRIPTION
     Performs the calculation *a = *a + b.
```

```
NAME
     IsAddTime - adds two time values

SYNOPSIS
     #include "Isutil.h"

     void IsAddTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Performs the calculation *a = *a + *b.
```

```
NAME
     IsClientConfig - get client configuration

SYNOPSIS
     #include "Isutil.h"

     long IsClientConfig(which)
     char *which;

ARGUMENTS
     which     Character string describing what client configuration item
               to return.

DESCRIPTION
     Returns the configuration string matching the specified description.

RETURN VALUE
     The requested configuration string if found, if not a NULL pointer is
     returned.

SEE ALSO
     IsServerConfig(3Is)
```

```
NAME
     IsCmpTime - compares two time values

SYNOPSIS
     #include "Isutil.h"

     int IsCmpTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Compares *a to *b.

RETURN VALUE
     Returns zero if *a == *b.  Returns 1 if *a > *b.  Returns -1 if *a <
     *b.
```

```
NAME
     IsDivTimeDouble - divides a time value with a float

SYNOPSIS
     #include "Isutil.h"

     void IsDivTimeDouble(a, seconds)
     IsTime *a;
     double seconds;

ARGUMENTS
     a         Pointer to an IsTime structure.

     seconds   Number of seconds to divide with.

DESCRIPTION
     Performs the calculation *a = *a / b.
```

```
NAME
     IsDivTime - divides two time values

SYNOPSIS
     #include "Isutil.h"

     void IsDivTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Performs the calculation *a = *a / *b.
```

```
NAME
     IsDouble2Time - converts floating point seconds to a time value

SYNOPSIS
     #include "Isutil.h"

     IsTime IsDouble2Time(seconds)
     double seconds;

ARGUMENTS
     seconds   Value to convert.

DESCRIPTION
     Converts a floating point value representing seconds to the internal
     time format.

RETURN VALUE
     The converted value.
```

```
NAME
     IsDumpCore - create a core dump of a running process

SYNOPSIS
     #include "Isutil.h"

     void IsDumpCore(name, pid)
     char *name;
     int pid;

ARGUMENTS
     name      Name of core file to dump.

     pid       Process id.

DESCRIPTION
     Creates a core dump of the specified process. The process will
     continue to run after the core file has been created.

NOTES
     Some systems doesn't provide a mechanism to dump a core of a running
     process, in that case this function just returns.  SunOS rejects
     attempts to dump core of a process that is attached to a debugger. If
     IsDumpCore(3Is) detects that the process is attached to a debugger it
     sends the SIGINT signal to the process.
```

```
NAME
     IsInt2Time - converts integer seconds to a time value

SYNOPSIS
     #include "Isutil.h"

     IsTime IsInt2Time(seconds)
     int seconds;

ARGUMENTS
     seconds   Value to convert.

DESCRIPTION
     Converts a integer value representing seconds to the internal time
     format.

RETURN VALUE
     The converted value.
```

```
NAME
     IsMjd2Time - convert mjd format to internal time format

SYNOPSIS
     #include "Isutil.h"

     void IsMjd2Time(mjd, ist)
     double mjd;
     IsTime *ist;

ARGUMENTS
     mjd       A modified julian day value. Number of days since Jan 1
               1950.

     ist       Pointer to an IsTime structure where the result is placed.

DESCRIPTION
     Converts from the modified julian day format to the internal time
     format.
```

```
NAME
     IsMulTimeDouble - multiplies a time value with a float

SYNOPSIS
     #include "Isutil.h"

     void IsMulTimeDouble(a, seconds)
     IsTime *a;
     double seconds;

ARGUMENTS
     a         Pointer to an IsTime structure.

     seconds   Number of seconds to multiply with.

DESCRIPTION
     Performs the calculation *a = *a * b.
```

```
NAME
     IsMulTime - multiplies two time values

SYNOPSIS
     #include "Isutil.h"

     void IsMulTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Performs the calculation *a = *a * *b.
```

```
NAME
     IsRetAddTime - adds two time values

SYNOPSIS
     #include "Isutil.h"

     IsTime IsRetAddTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Calculates *a + *b.

RETURN VALUE
     The result of the operation is returned.
```

```
NAME
     IsRetDivTime - divides two time values

SYNOPSIS
     #include "Isutil.h"

     IsTime IsRetDivTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Calculates *a / *b.

RETURN VALUE
     The result of the operation is returned.
```

```
NAME
     IsRetMulTime - multiplies two time values

SYNOPSIS
     #include "Isutil.h"

     IsTime IsRetMulTime(a, b)
     IsTime *a;
     IsTime *b;

ARGUMENTS
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.

DESCRIPTION
     Calculates *a * *b.

RETURN VALUE
     The result of the operation is returned.
```

NAME

```
     IsRetSubTime - subtracts two time values
```

SYNOPSIS

```
     #include "Isutil.h"

     IsTime IsRetSubTime(a, b)
     IsTime *a;
     IsTime *b;
```

ARGUMENTS

```
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.
```

DESCRIPTION

```
     Calculates *a - *b.
```

RETURN VALUE

```
     The result of the operation is returned.
```

```
NAME
     IsSeconds2Time - convert a seconds string to the internal time format

SYNOPSIS
     #include "Isutil.h"

     void IsSeconds2Time(str, ist)
     char *str;
     IsTime *ist;

ARGUMENTS
     str       Pointer to a character array holding the seconds string.

     ist       Pointer to IsTime structure where the result is placed.

DESCRIPTION
     A string of format "s.s" is converted to the internal time format.
```

```
NAME
     IsServerConfig - get client configuration

SYNOPSIS
     #include "Isutil.h"

     long IsServerConfig(which)
     char *which;

ARGUMENTS
     which     Character string describing what server configuration item
               to return.

DESCRIPTION
     Returns the configuration string matching the specified description.

RETURN VALUE
     The requested configuration string if found, if not a NULL pointer is
     returned.

SEE ALSO
     IsClientConfig(3Is)
```

```
NAME
     IsSubTimeDouble - subtracts a float from a time value

SYNOPSIS
     #include "Isutil.h"

     void IsSubTimeDouble(a, seconds)
     IsTime *a;
     double seconds;

ARGUMENTS
     a         Pointer to an IsTime structure.

     seconds   Number of seconds to subtract.

DESCRIPTION
     Performs the calculation *a = *a - b.
```

NAME
```
     IsSubTime - subtracts two time values
```

SYNOPSIS
```
     #include "Isutil.h"

     void IsSubTime(a, b)
     IsTime *a;
     IsTime *b;
```

ARGUMENTS
```
     a         Pointer to an IsTime structure.

     b         Pointer to an IsTime structure.
```

DESCRIPTION
```
     Performs the calculation *a = *a - *b.
```

```
NAME
     IsTime2Double - converts time value to floating point seconds

SYNOPSIS
     #include "Isutil.h"

     double IsTime2Double(t)
     IsTime *t;

ARGUMENTS
     t         Value to convert.

DESCRIPTION
     Converts the internal time format to a floating point value
     representing seconds.

RETURN VALUE
     The converted value.
```

```
NAME
     IsTime2Hms - convert internal time format to a string

SYNOPSIS
     #include "Isutil.h"

     void IsTime2Hms(ist, str)
     IsTime *ist;
     char *str;

ARGUMENTS
     ist       Pointer to an IsTime structure holding the time to be
               converted.

     str       Pointer to a character array that must be at last
               IsYMD_HMS_LEN characters long to hold the result.

DESCRIPTION
     Converts from the internal time format to a string of format
     "hhmmss.s".
```

```
NAME
     IsTime2Mjd - convert internal time format to mjd format

SYNOPSIS
     #include "Isutil.h"

     double IsTime2Mjd(ist)
     IsTime *ist;

ARGUMENTS
     ist       Pointer to an IsTime structure holding the time to be
               converted.

DESCRIPTION
     Converts from the internal time format to modified julian day format.

RETURN VALUE
     Number of days since Jan 1  1950.
```

```
NAME
     IsTime2Seconds - convert internal time format to a string

SYNOPSIS
     #include "Isutil.h"

     void IsTime2Seconds(ist, str)
     IsTime *ist;
     char *str;

ARGUMENTS
     ist       Pointer to an IsTime structure holding the time to be
               converted.

     str       Pointer to a character array that must be at last
               IsYMD_HMS_LEN characters long to hold the result.

DESCRIPTION
     Converts from the internal time format to a string of format "s.s".
```

```
NAME
     IsTime2VikStw - convert internal time format to Viking satellite time
     word

SYNOPSIS
     #include "Isutil.h"

     double IsTime2VikStw(ist)
     IsTime *ist;

ARGUMENTS
     ist       Pointer to an IsTime structure holding the time to be
               converted.

DESCRIPTION
     Converts from the internal time format to the Viking satellite time
     word.

RETURN VALUE
     Viking satellite time word.
```

```
NAME
     IsTime2YmdHms - convert internal time format to a string

SYNOPSIS
     #include "Isutil.h"

     void IsTime2YmdHms(ist, str)
     IsTime *ist;
     char *str;

ARGUMENTS
     ist       Pointer to an IsTime structure holding the time to be
               converted.

     str       Pointer to a character array that must be at last
               IsYMD_HMS_LEN characters long to hold the result.

DESCRIPTION
     Converts from the internal time format to a string of format "yymmdd
     hhmmss.s".
```

```
NAME
     IsTimeGm - convert a tm structure to unix seconds format

SYNOPSIS
     #include "Isutil.h"

     long IsTimeGm(t)
     struct tm *t;

ARGUMENTS
     t         A pointer to a tm structure holding the time to be
               converted.

DESCRIPTION
     Convert a tm structure to number of seconds since Jan 1  1970, the
     time is assumed to be in UT.  This function is identical to the POSIX
     mktime(3) and Sun timegm() functions.

RETURN VALUE
     Number of seconds since Jan 1  1970.
```

```
NAME
     IsutilInitialize - initialize the library

SYNOPSIS
     #include "Isutil.h"

     void IsutilInitialize(argc, argv)
     int argc;
     char **argv;

ARGUMENTS
     argc      Number of arguments. Same as argc in main().

     argv      Pointer to a table of argument strings.  Same as argv in
               main().

DESCRIPTION
     Initializes the Isutil library. It will also set the timezone for the
     program to UT (TZ=UTC).
```

```
NAME
     IsVikStw2Time - convert Viking satellite time word to internal time
     format

SYNOPSIS
     #include "Isutil.h"

     void IsVikStw2Time(orbit, stw, ist)
     int orbit;
     unsigned int stw;
     IsTime *ist;

ARGUMENTS
     orbit     The orbit number. Nescessary beacuse stw wraps around
               several times during the Viking life time.

     stw       Viking satellite time word.

     ist       Pointer to an IsTime structure where the result is placed.

DESCRIPTION
     Converts a Viking satellite time word to the internal time format.
```

```
NAME
     IsYmdHms2Time - convert a string to the internal time format

SYNOPSIS
     #include "Isutil.h"

     void IsYmdHms2Time(str, ist)
     char *str;
     IsTime *ist;

ARGUMENTS
     str       Pointer to a character array holding the time string.

     ist       Pointer to IsTime structure where the result is placed.

DESCRIPTION
     A string of format "yymmdd hhmmss.s" or "yymmdd-hhmmss.s" is converted
     to the internal time format.
```

```
NAME
     Vmalloc, VaVmalloc, Vnormalize, Vrelocate, Vsplice  - Matrix memory
     allocation routines

SYNOPSIS
     #include <Vmalloc.h>

     void *Vmalloc(unsigned size, unsigned dim, unsigned dims[],
                   unsigned *bytes);

     void *VaVmalloc(unsigned size, unsigned dim, ...);

     void Vnormalize(void *data, unsigned dim, unsigned dims[]);

     void Vrelocate(void *data, unsigned dim, unsigned dims[]);

     unsigned Vsplice(void *data, unsigned size, unsigned dim,
                   unsigned dims[], unsigned offset);


DESCRIPTION
     malloc allocates space for a matrix of dimension dim.  The sizes of
     the dimensions are in the array dims[]. dims[0] is the major dimension
     and dims[dim-1] is the minor dimension.  If bytes is non-NULL malloc
     will store the actual number of bytes that was allocated in the
     variable pointed to by bytes.  The space is allocated with a call to
     malloc(3), and it's the callers responsibility to free the space when
     it is no longer needed.

     The main advantages of malloc are that it calls malloc(3) only once,
     and that the allocated space is freed by a single call to free(3).

     aVmalloc is like malloc except that the dimensions are described in
     a varargs(3)/stdarg(3) fashion. The last argument must be a pointer to
     an unsigned int variable, or NULL.  This variable corresponds to bytes
     for malloc.  See also WARNINGS below.

     Both malloc and aVmalloc returns a pointer to the allocated space or
     NULL upon error.

     normalize normalizes the pointer structure of data (assumed to have
     been obtained by a call to malloc or aVmalloc) so that the space
     pointed to by data can be transported, copied, stored on disk, or
     whatever. See also BUGS below.

     relocate relocates the pointer structure of data (assumed to have
     been obtained by a call to malloc or aVmalloc) after a call to
     normalize has been made, so that the space pointed to by data is
     again usable as a matrix in C.  See also BUGS below.
     splice splices the matrix pointed to by data with respect to the
     major dimension. The data is shifted ‘upwards’ so that the maximum
     index for the major dimension is reduced by offset.  splice returns
     the number of bytes that makes up the new matrix, or 0 (zero) upon
     error (such as incorrect parameters).  Note that splice doesn't free
     up any allocated space.
```

EXAMPLES
```
     /* Allocate an integer matrix that is 2x3x4 in size and a short matrix
     that is 6x4x2x3. */
     unsigned bytes;
     unsigned dims[3] = {2, 3, 4};
     int ***ix2;
     int ***ix = (int ***)Vmalloc(sizeof(int), 3, dims, &bytes);
     short ****sx = (short ****)VaVmalloc(sizeof(short), 4, 6, 4, 2, 3, NULL);

     /* Assignment is like this:  */
     ix[1][2][3] = 20;
     sx[3][2][1][0] = 63;

     /* Copy the matrix ix to ix2. */
     ix2 = (int ***)malloc(bytes);
     Vnormalize(ix, 3, dims);
     memcpy(ix2, ix, bytes);
     Vrelocate(ix2, 3, dims);

     /* Splice the matrix ix2. */
     ix2[1][2][0] = 123456;
     bytes = Vsplice(ix2, sizeof(int), 3, dims, 1);

     /* This is now true (see assignment to ix above) */
     if (ix2[0][2][0] == 123456 && ix2[0][2][3] == 20) ...

     /* Free up space. */
     free(sx);
     free(ix);
     free(ix2);
```

AUTHOR
     Jan D. <jhd@irfu.se>

WARNINGS
     aVmalloc can't handle matrixes with more than 10 dimensions.  Use
     malloc if such matrixes are needed.

BUGS
     The size of pointers may be different on different computers. When
     compiling these routines you decide how many bytes a pointer at most
     will occupy (typically 4 or 8). If you specify 8, it can also handle
     any size less than 8. However, this routines will only work together
     if they have been compiled with the same value.
     When choosing size, beware that most computers requires pointers to be
     aligned on an 4 byte boundary. A value like 3 or 6 will probably give
     you a bus error.

SEE ALSO
     free(3) malloc(3) stdarg(3) varargs(3)