CSDS User Interface ISDAT User Defined C clients

Swedish Institute of Space Physics, Uppsala Division

with change bars for changes introduced in issue 2.0

Contents

1 Introduction

1.1 Intended readership

This manual is intended for the user of the CSDS User Interface ISDAT Client package who wants to design and add his personal clients written in the C language.

1.2 Applicability of the manual

The current version of the document applies to the ISDAT version 2.2, delivered as release 4 within the CSDS User Interface Project. It is valid for UNIX, SUN Solaris workstations.

1.3 Purpose of the software

The purpose of the CSDS User Interface Data Manipulation software package is to provide the scientific community with software tools to manipulate and display Cluster CSDS summary and primary parameters. The writing of user defined C clients provides a means to add local, personal and special purpose software modules for data manipulation and display.

1.4 How to use this document

This document basically consist of comments to the source code of a simple C client to be included in an ISDAT client package. The intention is that the reader from this should be able to write his own C clients by using this as a model, or simply make modifications and additions to the model program in order to meet his personal needs.

1.5 Related documents

An overview of the CSDS UI ISDAT Client Package is given in [Ref. 3]. It is assumed that the reader is familiar with the information given in that manual. Instructions for adding user defined clients written in IDL language are given in [Ref. 4].

1.6 Conventions and acronyms

In this manual, we will use:

- *italics* to indicate exact names or expressions.
- Courier fonts to give command line expressions or source code.
- > to indicate the terminal prompter.

Acronyms and abbrieviations used are described in Table 1.

Acronym	Meaning
CSDS	Cluster Science data System
CUI	CSDS User Interface
IDL	Interactive Data Language
IRF-U	Institutet för Rymdfysik, Uppsalaavdelningen
	Swedish Inst. of Space Phys., Uppsala Division
ISDAT	Interactive Science Data Analysis Tool
TBD	To be defined
TBW	To be written
UI	User Interface

Table 1: Acronyms and abbrieviations

1.7 Reference Documents

[1] CSDS User Interface ISDAT Architectural Design Document. Technical Report DS-IRF-AD-0001, IRF-U, September 1995. Issue 1.0.

[2] CSDS User Interface, ISDAT Installation Manual. Technical Report DS-IRF-IM-0001, IRF-U, September 1995.

[3] CSDS User Interface, ISDAT User Manual. Technical Report DS-IRF-UM-0001, IRF-U, September 1995.

[4] CSDS User Interface, ISDAT search Client User's Manual. Technical Report DS-IRF-UM-0008, IRF-U, September 1995.

[5] B. W. Kernighan and D. M. Ritchie. *The C Programming Language, ANSI C*. Prentice Hall, 1988.

[6] WECdata structure working group. Editor C. Harvey. The structure of the WEC/ISDAT data. Technical Report CWD-OBSPM-DD-001, OBSPM, March 1995.

1.8 Problem reporting

Problems should be reported to the CSDS National Data Centre.

2 Overview

2.1 Prerequisites and assumptions

For these instructions, it is assumed that you are familiar with C and the C syntax. If that is not the case, please consult your C manuals, for example [Ref. 5]. It is also assumed that your local work station has C properly installed and that you have ISDAT installed at your local workstation.

If you intend to write your own local client you might want to learn more about the internal structure of the ISDAT system, although it is not necessary. More information about the architecture is found in the CSDS UI ISDAT Architectural Design Document [Ref. 1]. It is also useful to acquire a better understanding of the data structures in the ISDAT client server communications by consulting [Ref. 6].

2.2 Development procedure and tools

2.2.1 Development steps

The inclusion of user defined clients in the CSDS User Interface ISDAT client package requires access to the full ISDAT client package source code. The procedure roughly should follow these steps:

1. Acquire the ISDAT CSDS UI ISDAT Client Package source code as described in section 2.2.2.
2. Install the source code and build the system as described in section 2.2.3.
3. Create a sub-directory for your local client as described in section 2.2.4.
4. Write your local code using the examples in section 3 as guides or templates, following the coding conventions proposed in section 2.4 and making use of the supporting libraries described in section 2.3.
5. Document your local client as described in section 3.1.6.
6. Integrate your local client with the ISDAT system as described in section 3.1.4.

2.2.2 How to acquire the ISDAT source code

Normally, only executable code is distributed to the CSDS users. In order to integrate a local client in the system you need to build the system from source code. Contact your CSDS National Data Centre for information on how to obtain the required source code and installation manuals.

2.2.3 How to build the ISDAT from source code

Complete information on the installation and building of the ISDAT from source code is found in [Ref. 2].

2.2.4 Creating a sub-directory for the local client

The top level of the ISDAT directory tree is:

```
/isdat
        /clients
        /man
        /server
        /local
        /
```

It is recommended to create a sub-directory under /isdat/local named after your clients:

```
/isdat/local/myclient1
            /myclient2
            /myclient3
```

2.3 ISDAT supporting libraries

The following supporting libraries are included in the ISDAT CSDS UI source code distribution:

Dblib for data base related functions. The Dblib calls are described in Appendix C.

Islib for general ISDAT calls. Islib calls are described in Appendix D.

Isutillib for ISDAT utility functions, time conversions etc. The Isutillib calls are described in Appendix E.

Tblib ISDAT toolbox. Consult the on-line man pages for details.

Molib Motif related functions. Consult the on-line man pages for details.

2.4 Coding conventions

The ISDAT code is following a few simple rules and conventions. It is recommended to follow those rules also for user defined clients in order to get uniform and understandable code within the whole system. For your guidance, the rules are included in the following sections.

2.4.1 Documentation standards

All C library calls and C executable processes shall have an associated on-line man-page following UNIX standards. All other C units shall be written in a self-documentary style including a standard ASCII header with the following information and syntax:

```
#MN  Module Name:
#MD  Module Description:
#MA  Module Author:

#IN  Include Name:
#ID  Include Description:
```

```
#FN  Function Name:
#FD  Function Description:
#FC  Function Constraints:
#FID Function Interface Description:
#FI  Function Input Parameter
#FIO Function Input/Output Parameter
#FO  Function Output Parameter
#FR  Function Returned Value
```

2.4.2 Naming conventions

The following naming conventions shall be used:

- Begin variable names by lower case, e.g. **variable**
- Indicate multi-word variables by upper case, e.g. **secondVariable**
- Begin functions by upper case, e.g. **ComputeAverage()**
- Use all-upper-case for define, e.g. **#define PI 3.14159** and underscore for multi-word names e.g. **#define PI_HALF 1.57**.

2.4.3 Programming standards

The following standards and rules should be applied:

- All units should follow ANSI C standard
- Group families of variables into *structures* to avoid long argument lists.
- Avoid long functions. Limit each function to one or two A4 pages as a rule. Otherwise split up into several functions.
- Propagate errors to the top level. Never print out errors in the low level functions.
- Favour readable and logic code before the fastest possible code.
- Never use hard coded paths in the code.

3 Local client code

The instructions on how to write a local ISDAT C client will use a client that prints out the maximum value of a parameter read from the data base. We will proceed in a two-step process where the first example, client cuiexx, will simply print out the value in a terminal window. This is described in section 3.1. Then we will, in section 3.2, add on a graphical user interface to the simple cuiexx client, re-name it to cuiex, make it appear on the *clients* menu ot the time manager, and accept user specification of the parameter to be used. The cuiex client will thus become a *general client* meaning that it will work independently of projects and instruments.

3.1 A very simple local client

We are now going to write a C client that will print out the maximum value of a hard-coded parameter within the specified time interval. We will name the client *cuiexx*. It will be a *special client* meaning that it will function only for a specific project or instrument (see [Ref. 3]). We start by writing a main program:

3.1.1 Main program

The function of the main process is to initialise the processing and then enter into the main loop and remain there pending callbacks from the time manager. The main call utilises the standard arguments argc and argv.

```
int main(argc, argv)
int argc;
char **argv;
{
}
```

which may be used to communicate information about the desired display or other flags. A description of useful flags is given in section 3.1.4. For example: *-display myscreen:0.* If no information is given, the display given by the environment variable $DISPLAY is used (consult your UNIX system manuals or your system manager if you are not familiar with environment variables).

The declarations needed are:

```
    int error;
    IsTmInfo *tmInfo;
    Display *display;
    Database *db;         /* pointer to open database */
```

The Display declaration is an X11 declaration and Database is ISDAT specific. They are defined in header files that have to be included. In most clients you will need defines given in the following include files:

```
#include <Isstd.h>
#include <Is.h> /* Isdat declarations */
#include <Db.h> /* Data base declarations */
```

They are ISDAT specific include files used by the IsLib and DbLib libraries. The ISDAT system do know where to find them so you do not need to specify the full path. The corresponding calls are described in the appropriate on line manuals and in appendicies of this manual.

The first step is to initialize the ISDAT libraries:

```
    IsInitialize(argc, argv, display);
```

where the *XOpenDisplay()* call is described in the X11 manuals. The *IsInitialize()* call is described in the IsLib on line man pages (Appendicix D, page 87).

Next, we try to open a connection to a data base handler (ISDAT server):

```
    if ((db = DbOpen(NULL,argc,argv)) == NULL) {
        printf("Cannot connect to database %s",DbName(NULL));
        exit(1);
    }
```

The DbOpen call is defined in the DbLib on line manual (See Appendix C page 59). In this example we try to connect to the data base handler defined by the environment variable $ISDAT_DATABASE. This is an ISDAT defined environment variable.

The final initialisation is to tell the *time manager* where to direct the time event callbacks:

```
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);
```

This call is described in the IsLib on line manual pages, Appendix D, page 75. In this example we direct the time callbacks to the function *UpdateTimeCB* (see below).

Now we have done all necessary initialisation, so we just have to enter the main loop and wait for callbacks from the time manager.

```
    IsMainLoop();
```

This call is described in the IsLib on line manual pages (Appendix D, page 88). We will now stay within the main loop until the end of the execution.

Our complete main process is thus:

```
#include <Isstd.h>
#include <Is.h> /* Isdat declarations */
#include <Db.h> /* Data base declarations */

Database *db; /* pointer to open database */

extern void UpdateTimeCB();

int main(argc, argv)
int argc;
char **argv;
{
    int error;
    IsTmInfo *tmInfo;
    Display *display;

    /* initialize isdat library */
    error = IsInitialize(argc, argv, display);
    if (error) {
        printf("Could not find a time manager\n");
        exit(1);
    }

    /* Connect to the data base handler */
    if ((db = DbOpen(NULL,argc,argv)) == NULL) {
        printf("Cannot connect to database %s",DbName(NULL));
        exit(1);
    }

    /* add callback for time manager information */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    /* enter main loop */
    IsMainLoop();
    exit(0);
}
```

3.1.2 Time event callbacks

In the main process, we told the connected time manager to send all time event callbacks to process *UpdateTimeCB()*. Time events are typically when the time manager update the time or change the time interval. When the time manager sends the callback, it also provides some information about the data that it can provide for the time interval in question. This is done via arguments to the call:

```
void UpdateTimeCB(reason, data, closure)
int reason; /* callback reason */
IsPointer data; /* orbit manager information */
IsPointer closure;
```

where *reason* is set to IsCR_TM_INFO when called as a result of a time manager action, *data* is a pointer to a structure described in the IsLib on line manual pages (Appendix D, *page* 94), and closure is the value of the third argument to *IsAddCallback* (NULL in our example). The complete callback function is:

```
void UpdateTimeCB(reason, data, closure)
int reason; /* callback reason */
IsPointer data; /* orbit manager information */
IsPointer closure;
{
    static IsTmInfo info;      /* time manager info */

    info = (IsTmInfo *)data;
    Update(info);               /* update result */
}
```

where we cast the returned information to *IsTmInfo* format (info = (IsTmInfo *)data), (see page 94). The information (in this case we need time interval) is forwarded to our *Update* function via the argument *info.* The *Update()* function is the function that will really do the job and that is the function where you would put your own processing. It is described in the following section.

3.1.3 Application code

Function *Update()* is used to update the result as a result of the time manager is changing the time interval or issuing an *event* for some other reason.

```
void Update(tmInfo)
IsTmInfo *tmInfo;
{
    int error;
    DbDataRequest req; /* data request */
    DbDataObject *obj; /* data object */
```

To start with we have to declare a data structure that will describe the data we are going to request from the ISDAT server and a structure which will contain the returneed data. The two declarations DbDataRequest and DbDataObject are described in Appendix C page 47 and the following pages. Note that no memory is allocated to *obj.* This is done by *DbGetData* and has to be freed by our program when we no longer need the data returned (see below).

Next, we have to describe the data we want. In ISDAT this is done by defining a *conceptual instrument* in a hierarchy consisting of the items *project member instrument sensor signal channel parameter.* Here we use the structure *req.* Usually, the client would query the ISDAT server about available conceptual instruments. However, in this case we will hard code the specification just to make this client more understandable. **It is not recommended to do that as a general practice**. In the CSDS application we have a problem to "hard code" the requested *virtual instrument* since the parameters are read from the CDF file and names and parameters are not at all hard coded into the ISDAT server. However, we can get around the problem by inspecting the names displayed in the selection menues of other clients (for example cuigr) and then convert the names to numbers by using the *DbName2Spec* function described on page 57. We declare a temporary structure

```
    DbSpecName name;
```

and use it to fill in the strings:

```
    strcpy(name.project,"CSDS_PP");
    strcpy(name.member,"C1");
    strcpy(name.instrument,"EFW");
    strcpy(name.sensor,"E_dusk");
    strcpy(name.channel,"");
    strcpy(name.parameter,"");
    DbName2Spec(db,&name,&req.spec);
```

We also have to specify the time interval, desired units, no data reduction, and that we want time tagged data:

```
    req.start = tmInfo->start;
    req.interval = tmInfo->interval;
    req.units = DbUN_PHYS;
    req.reduction = DbRED_NONE;
    req.pack = DbPACK_TIMETAG;
```

where the times have been obtained from the time manager via the *info* structure as described above.

Now we are ready to request the data from the ISDAT server (see page 47 for a description of the function *DbGetData*):

```
    error = DbGetData(db, &req, &obj);
```

and make sure that there are no errors:

```
   if (error == DbSUCCESS) {
        /* check if returned data is of right format */
        if (obj->rank == 0 && obj->dataType == DbTYPE_REAL_FLOAT &&
            obj->dimension == 0) {
            printf("max: %g %s\n", GetMax(obj), GetUnits(obj));
        } else {
            printf( "Can't handle\nrank = %d\ntype = %d\ndimension = %d",
                    obj->rank, obj->dataType, obj->dimension);
        }
        DbFreeDataObject(obj);
    } else {
        printf("Error: %s\n", DbErrorString(error));
    }
```

If there are errors we print out the returned error strings by using the function*DbErrorString()* described on page 42. If the call was successful, we have to check for the variable type since our simple client only knows how to handle real scalars of rank 0 and dimension 0.

After we are ready with the use of the data it is important to free the memory allocated to the data by using the function *DbFreeDataObject()*. DbFreeDataObject() is described on page 43.

If all is OK we print out the result including the units by utililizing the GetMax and GetUnits functions described below. The complete Update() function is thus:

```
#include <Isstd.h>
#include <Is.h>
#include <Ui.h>
#include <Db.h> /* Data base declarations */

static double GetMax();
static char *GetUnits();
```

```
DbDataObject *obj;
extern Database *db;

void Update(tmInfo)
IsTmInfo *tmInfo;
{
    int error;
    DbDataRequest req; /* data request */
    DbDataObject *obj; /* data object */
    DbSpecName name;
    char message[256];
    /* Arg args[20]; */

    /* describe the data we want */
    req.start = tmInfo->start;
    req.interval = tmInfo->interval;
    req.units = DbUN_PHYS;
    req.reduction = DbRED_NONE;
    req.pack = DbPACK_TIMETAG;
    strcpy(name.project,"CSDS_PP");
    strcpy(name.member,"C1");
    strcpy(name.instrument,"EFW");
    strcpy(name.sensor,"E_dusk");
    strcpy(name.channel,"");
    strcpy(name.parameter,"");
    DbName2Spec(db,&name,&req.spec);

    /* get the data from the database handler */
    error = DbGetData(db, &req, &obj);

    /* check if any errors */
    if (error == DbSUCCESS) {
        /* check if returned data is of right format */
        if (obj->rank == 0 && obj->dataType == DbTYPE_REAL_FLOAT &&
            obj->dimension == 0) {
            printf("max: %g %s\n", GetMax(obj), GetUnits(obj));
        } else {
            printf( "Can't handle\nrank = %d\ntype = %d\ndimension = %d",
                    obj->rank, obj->dataType, obj->dimension);
        }
        DbFreeDataObject(obj);
    } else {
        printf("Error: %s\n", DbErrorString(error));
    }
}
```

Now we just have to actually identify the maximum value. The code for the GetMax() function is:

```
static double GetMax(obj)
DbDataObject *obj;
{
    int i;
    int seg;
    double max;
    DbDataRF0 *rf0;

    /* set initial max value */
    rf0 = (DbDataRF0 *)obj->seg[0].data;
    max = rf0[0].re;

    /* loop over all samples looking for maximum value */
    for (seg = 0; seg < obj->segments; seg++) {
        rf0 = (DbDataRF0 *)obj->seg[seg].data;
        for (i = 0; i < obj->seg[seg].samples; i++) {
            if (rf0[i].re > max) max = rf0[i].re;
        }
    }
    return max;
}
```

We are dealing with only real float variables of rank 0. Therefore we declare a a pointer to a variable matching this by the *DbDataRF0 *rf0* statement. See [Ref. 6] for more details on this declaration.

We then cast our data to this type whenever used. In general, the data may contain data gaps. Therefore the ISDAT server returns data in segmented form containing several segments of contiguous data. We therefore have to use a loop over all returned data segments when we search for the maximum value. After having completed the loop, we return the identified max value.

We also want to know the units of our quantity. The units are supplied with the returned data object. We use the function *GetUnits()* to return the units to *Update()*:

```
static char *GetUnits(obj)
DbDataObject *obj;
{
    return obj->info[0].unitString;
}
```

Now we have completed the coding of our very simple client. The complete code is listed in Appendix A. The source code is included in the CSDS distribution under directory *demo*. We now have to compile, load, and integrate our *cuiexx* client with the rest of the ISDAT syystem. This is explained in section 3.1.4 and 3.1.5.

3.1.4 Compiling and loading the very simple client

ISDAT uses the *make* tool to compile and link the components. *make* gets its instructions from a special file called *Makefile.* In ISDAT, the whole system or a single component (i.e. a client) can be built by one *make* command depending on where in the tree it is given. For example, a command

```
 >make
```

in the top directory will build the complete ISDAT system, while the same command given in your local client directory will only build your local client. However, in order to make the ISDAT installation platform independent, another tool *Imake* is used to actually create the Makefiles from files called *Imakefile.* Therefore **you should never edit or import a Makefile but only the Imakefile**. The Imakefile for our *cuiexx* client looks like this:

```
/*
   Imake template for cuiexx. Imake will generate a Makefile
   from this template.
   To regenerate the Makefile after a change to Imakefile type:
   make Makefile (or imkmf).
*/

        DEPLIBS = $(DEPUILIB) $(DEPDBLIB) $(DEPISLIB) $(DEPISUTILLIB) \
                  $(DEPMOLIB)
LOCAL_LIBRARIES = $(UILIB) $(DBLIB) $(ISLIB) $(ISUTILLIB) \
                  $(MOLIB) \
                  $(XMLIB) $(XTLIB) $(X11LIB) \
                  $(MATHLIB)
       INCLUDES = -I. $(XMINC) $(XTINC) $(X11INC)

           SRCS = main.c cb.c update.c
           OBJS = main.o cb.o update.o

ComplexProgramTarget(cuiexx)
RegisterClient(cuiexx,demo,cuiexx)
```

where we have specified libraries, include files, source files, and object files. The last two lines are ISDAT specific and specify the client name and how the client should be registred at the *time manager* that eventually adopts the client in its family of clients. In this case, we tell the time manager to include our *cuiexx* client under the name cuiexx under the *clients* menu and under the *demo* sub-menu. For a real application case, it would be more appropriate to use *csds* instead of *demo.*

When you have created the *Imakefile* in the directory the */isdat/local/cuiexx* (replace cuiexx by your preferred name of the client), you should do:

```
>cd ..
>make Makefiles
>cd cuiexx
>make
```

That is, we have assumed that we start from the local client directory *cuiexx* and first change the current directory to */isdat/local.* In that directory there is already a pre-prepared *Imakefile* and a *Makefile.* The command *make Makefiles* means that we update the *Makefile* of the *local* directory and create a *Makefile* in the sub-directories, including our *cuiexx* directory. Note the "s" in the "Makefiles. Then ve go back to our *cuiexx* directory and give the *make* command to actually compile, link and registrer our client. Note that there are several targets that can be used with the *make* command like *make clean* etc.

After you have been through this cycle once, you do not need to run the command *make Makefiles* unless you re-configure your system. Just type

```
> make
```

when you want to re-compile your program.

3.1.5 Running the very simple client

A client can be run from any normal terminal window on the workstation by just typing the name of the program, in our case:

```
 >cuiexx
```

. *cuiexx* then "looks around" for running *time managers* and attaches itself to the first active time manager that is found.

Our simple client only knows how to print the result on the standard output, i.e. the terminal window. If it had been started from a terminal screen, that had been no problem. When started from the time manager, it will probably present the result on the terminal window from which the CSDS UI Session Manager was started.

Our simple client has no nice mechanism to exit. It will however, be terminated when the *time manager* is terminated.

3.1.6 Documentation of the very simple client

Each ISDAT client should be accompanied by a *man page* with the standard UNIX man pages layout. It should be written in *Nroff* language. If the local ISDAT system is properly set up it should be possible to see any ISDAT related man-page by giving the command:

```
 >man clientname
```

For the *cuiexx* client the man source file *cuiexx.1* might look:

```
.TH cuiexx 1 "1.0" "ISDAT" "client"
.SH NAME
cuiexx \- displays max values for parameter CSDS_PP-C1-efw-E_dusk
.SH SYNOPSIS
cuiexx
```

```
.SH ARGUMENTS
Handles all generic ISDAT and X arguments.
.SH DESCRIPTION
.PP
Cuimeta is an ISDAT client of type "special clients". It is
designed for CSDS prime (CSDS-PP)
parameter data bases. It is intended just for demonstration
on how to write user defind C clients

.\".SH FILES
.\".SH NOTES
.SH SEE ALSO
cuitm.1 cuiex.1
.\".SH BUGS
.\".SH WARNINGS
```

In order to look at the formatted output do:

```
>nroff -man cuiexx.1
```

and the formatted page should appear on the standard output. In our example the formatted result would look:

```
 NAME
      cuiexx - displays max values for parameter CSDS_PP-C1-efw-E_dusk

 SYNOPSIS
      cuiexx

 ARGUMENTS
      Handles all generic ISDAT and X arguments.

 DESCRIPTION
     Cuimeta is an ISDAT client of type "special clients". It is designed
     for CSDS prime (CSDS-PP) parameter data bases. It is intended just for
     demonstration on how to write user defind C clients

 SEE ALSO
     cuitm.1 cuiex.1
```

3.1.7 Hints for modifications

The very simple client *cuiexx* is not the type of client that normally would satisfy our needs. For example we have no controlled way of exiting from the client. Normally we need a user interface and a graphical display. There is nothing that prevents you from using your normal graphics systems and libraries provided that you update the Imakefile accordingly. In other ISDAT applications (i.e. non-CSDS) XGKS, as well as PHIGS/PEX graphics clients are in use. The CSDS provided *cuigr* client is coded directly in X/Motif. Most of the other CSDS provided client interfaces are created using the X-designer tool.

In order to add your own interfaces, you would normally replace and expand the cuiexx update() function and add the necessary initializations in the main program.

In section 3.2 we will show how to transform the *cuiexx* client to a client of class *general clients* and we will add a user interface based on X11/Motif.

3.2 Expansion of the very simple client

This section assumes that you are familiar with section 3.1.

In this example we are going to expand the *cuiexx* client to have a user interface based on X11/Motif. We will not describe the details of the user interface. However, it is included in order to make it possible to include a menu bar that will allow you to specify any available parameter in the data base. In this way the client will become a *general client* i.e independent of the project and instrument.

All Xt calls are described in the relevant X manuals. Mo*** calls are ISDAT Motif related calls, described in the ISDAT on line manuals. Tb calls are ISDAT toolbox calls described in the ISDAT Tblib on line manuals. Neither of these calls are described in detail in theis example.

The complete *cuiex* source code is listed in Appendix B.

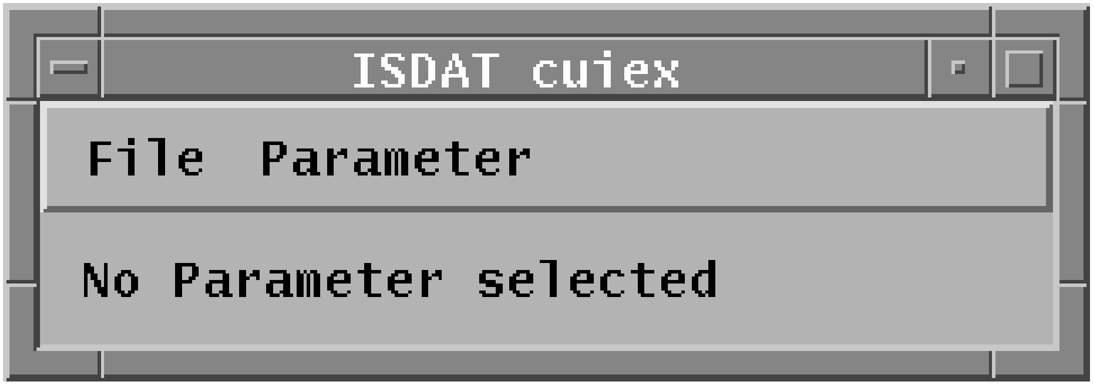
3.2.1 Main program

In the main process we first initialize the *Xt toolkit* needed for the X/Motif user interface by:

```
XtToolkitInitialize();
app_context = XtCreateApplicationContext();
dpy = XtOpenDisplay(app_context, NULL, argv[0], "Cuiex",
                    NULL, 0, &argc, argv);
if (!dpy) {
    printf("%s: can't open X window display\n", argv[0]);
    exit(1);
}
```

where we have also opened the display. The variables have been initialized as global variables:

```
int error;
XtAppContext app_context;
Display *dpy;
IsTmInfo *tmInfo;
MoMenuElement *paramHandle;
char message[256];
```



No Parameter selected

Figure 1: Client cuiex window

together with some other variables that we need later. we will not explain the Xt functions here. The new interesting feature of the main process is:

```
/* create the parameter menu */
specValid = 0;
dataSpec.project = tmInfo->project;
dataSpec.member = tmInfo->member;
dataSpec.instrument = tmInfo->instrument;
paramHandle = TbCreateSpecMenu(paramMenuW, db, &dataSpec,
                               DbLEVEL_SENSOR, DbLEVEL_PARAMETER, ParamCB);
if (!paramHandle->next) {
     MoMessageDialog(shellW, MoDIALOG_OK, "ISDAT cuiex connect",
    "No data for this\n\
     Project-Member-Instrument specification,\n\
     Terminate cuiex"
    , 0);
    exit(1);
}
```

where our client queers the ISDAT server about the available data description hierarchy and interactively builds a menu to be included in the client user interface.

3.2.2 Time event callbacks

There are no changes in the *UpdateTimeCB()* function.

3.2.3 Application code

In the *Update()* function we notice that we no longer "hard code" the data description hierarchy. Instead there are new callbacks *ParamCB* that dynamically updates the *Parameter* menu. We have also added a *Quit()* function that is called when the user presses the *Exit* button.

3.2.4 Graphical user interface

The function *CreateShellW()* includes no ISDAT specific parts. It is not explained in this manual. Please consult an X11 manual.

The final user interface is shown in Figure 1.

A Local ISDAT C client cuiexx source code list

A.1 File main.c

```
#include <Isstd.h>
#include <Is.h> /* Isdat declarations */
#include <Db.h> /* Data base declarations */

Database *db; /* pointer to open database */

extern void UpdateTimeCB();

int main(argc, argv)
int argc;
char **argv;
{
    int error;
    IsTmInfo *tmInfo;
    Display *display;

    /* initialize isdat library */
    error = IsInitialize(argc, argv, display);
    if (error) {
        printf("Could not find a time manager\n");
        exit(1);
    }

    /* Connect to the data base handler */
    if ((db = DbOpen(NULL,argc,argv)) == NULL) {
        printf("Cannot connect to database %s",DbName(NULL));
        exit(1);
    }

    /* add callback for time manager information */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    /* enter main loop */
    IsMainLoop();
    exit(0);
}
```

A.2 File cb.c

```
#include <Isstd.h>
#include <Is.h>
#include <Db.h> /* Data base declarations */


void UpdateTimeCB(reason, data, closure)
int reason; /* callback reason */
IsPointer data; /* orbit manager information */
IsPointer closure;
{
    IsTmInfo *info;

    info = (IsTmInfo *)data;
    Update(info);      /* update window */
}
```

A.3 File update.c

```
#include <Isstd.h>
#include <Is.h>
#include <Ui.h>
#include <Db.h> /* Data base declarations */

static double GetMax();
static char *GetUnits();

DbDataObject *obj;
extern Database *db;

void Update(tmInfo)
IsTmInfo *tmInfo;
{
    int error;
    DbDataRequest req; /* data request */
    DbDataObject *obj; /* data object */
    DbSpecName name;
    char message[256];
    /* Arg args[20]; */

    /* describe the data we want */
    req.start = tmInfo->start;
    req.interval = tmInfo->interval;
    req.units = DbUN_PHYS;
    req.reduction = DbRED_NONE;
    req.pack = DbPACK_TIMETAG;
    strcpy(name.project,"CSDS_PP");
    strcpy(name.member,"C1");
    strcpy(name.instrument,"EFW");
    strcpy(name.sensor,"E_dusk");
    strcpy(name.channel,"");
    strcpy(name.parameter,"");
    DbName2Spec(db,&name,&req.spec);

    /* get the data from the database handler */
    error = DbGetData(db, &req, &obj);

    /* check if any errors */
    if (error == DbSUCCESS) {
        /* check if returned data is of right format */
        if (obj->rank == 0 && obj->dataType == DbTYPE_REAL_FLOAT &&
            obj->dimension == 0) {
            printf("max: %g %s\n", GetMax(obj), GetUnits(obj));
        } else {
            printf( "Can't handle\nrank = %d\ntype = %d\ndimension = %d",
                    obj->rank, obj->dataType, obj->dimension);
        }
        DbFreeDataObject(obj);
    } else {
        printf("Error: %s\n", DbErrorString(error));
    }
}
```

```
static double GetMax(obj)
DbDataObject *obj;
{
    int i;
    int seg;
    double max;
    DbDataRF0 *rf0;

    /* set initial max value */
    rf0 = (DbDataRF0 *)obj->seg[0].data;
    max = rf0[0].re;

    /* loop over all samples looking for maximum value */
    for (seg = 0; seg < obj->segments; seg++) {
        rf0 = (DbDataRF0 *)obj->seg[seg].data;
        for (i = 0; i < obj->seg[seg].samples; i++) {
            if (rf0[i].re > max) max = rf0[i].re;
        }
    }
    return max;
}

static char *GetUnits(obj)
DbDataObject *obj;
{
    return obj->info[0].unitString;
}
```

B Local ISDAT C client cuiex source code list

B.1 File main.c

```
/*
#MN Module name: $Id: main.c,v 1.1 1995/03/08 18:24:34 al Exp al $
#MA Anders Lundgren, IRF-U

#MD Module description:
#MD The main program for the cuiex client.
#MD The client calculates the maximum value of the
#MD specified parameter during the requested interval.
*/

#include <Isstd.h>
#include <Is.h> /* Isdat declarations */
#include <Db.h> /* Data base declarations */
#include <Mo.h>
#include <Tb.h>
#include "gui.h"

static char *rcsId = "$Id: main.c,v 1.1 1995/03/08 18:24:34 al Exp al $";

int specValid; /* set if dataSpec is valid */
Database *db; /* pointer to open database */
DbDataSpec dataSpec; /* selected data specification */

extern void GetStat ();
extern void ParamCB();
extern void UpdateTimeCB();


/*
#FN Function name: main
#FD Function description:
#FD Main function.

#FID Interface:
#FI argc
#FI argv
*/
int main(argc, argv)
int argc;
char **argv;
{
    int error;
    XtAppContext app_context;
    Display *dpy;
    IsTmInfo *tmInfo;
    MoMenuElement *paramHandle;
    char message[256];

    /* initialize Xt toolkit and open display */
    XtToolkitInitialize();
    app_context = XtCreateApplicationContext();
    dpy = XtOpenDisplay(app_context, NULL, argv[0], "Cuiex",
NULL, 0, &argc, argv);
    if (!dpy) {
        printf("%s: can't open X window display\n", argv[0]);
        exit(1);
    }

    /* create graphical user interface */
    CreateShellW(dpy, argv[0], argc, argv);

    /* initialize isdat library */
    error = IsInitialize(argc, argv, dpy);
    if (error) {
        /* couldn't find a time manager on the display */
        MoMessageDialog(shellW, MoDIALOG_OK, "ISDAT cuiex connect",
        "no time manager, terminate cuiex", 0);
         exit(1);
    }

    /* Connect to the data base handler */
    if ((db = DbOpen(NULL,argc,argv)) == NULL) {
        sprintf(message,
                "cannot connect to database %s, terminate cuiex",
                DbName(NULL));
        MoMessageDialog(shellW, MoDIALOG_OK, "ISDAT cuiex connect",
                        message, 0);
        exit(1);
    }

    /* get which project, member and instrument to use */
    tmInfo = IsGetTmInfo();

    /* create the parameter menu */
    specValid = 0;
    dataSpec.project = tmInfo->project;
    dataSpec.member = tmInfo->member;
    dataSpec.instrument = tmInfo->instrument;
    paramHandle = TbCreateSpecMenu(paramMenuW, db, &dataSpec,
                                   DbLEVEL_SENSOR, DbLEVEL_PARAMETER, ParamCB);
    if (!paramHandle->next) {
        MoMessageDialog(shellW, MoDIALOG_OK, "ISDAT cuiex connect",
        "No data for this\n\
         Project-Member-Instrument specification,\n\
         Terminate cuiex"
          0);
        exit(1);
    }

    /* realize the GUI */
    XtRealizeWidget(shellW);

    /* add callback for time manager information */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    /* enter main loop */
    IsMainLoop();
    exit(0);
}
```

B.2 File gui.h

```
/*
#IN Include name: $Id: gui.h,v 1.1 1995/03/08 18:24:34 al Exp al $
#IA Anders Lundgren, IRF-U

#ID Include description:
#ID External definitions for the GUI.
*/

extern Widget shellW;
extern Widget paramCascadeW;
extern Widget paramMenuW;
extern Widget labelW;

extern void CreateShellW(Display *, char *, int, char **);

}
```

B.3 File gui.c

```
/*
#MN Module name: $Id: gui.c,v 1.1 1995/03/08 18:24:34 al Exp al $
#MA Anders Lundgren, IRF-U

#MD Module description:
#MD The graphical user interface.
*/

#include <Xm/Xm.h>
#include <Xm/CascadeB.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>


static char *rcsId = "$Id: gui.c,v 1.1 1995/03/08 18:24:34 al Exp al $";

Widget shellW;
Widget paramMenuW;
Widget labelW;

extern void Quit(Widget, XtPointer, XtPointer);


/*
#FN Function name: CreateShellW
#FD Function description:
#FD Creates the graphical user interface.

#FID Interface:
#FI display - pointer to opened display
#FI appName - the application name
#FI argc - argc from main()
#FI argv - argv from main()
*/
void CreateShellW(Display *display, char *appName, int argc, char **argv)
{
    int n; /* Arg Count */
    Arg args[20]; /* Arg List */
    XmString xmstr; /* temporary storage for XmStrings */
    Widget rowcolW;
    Widget menubarW;
    Widget fileCascadeW;
    Widget filePulldownW;
    Widget paramCascadeW;
    Widget exitButtonW;

    /* create application shell */
    n = 0;
    XtSetArg(args[n], XmNallowShellResize, TRUE); n++;
    XtSetArg(args[n], XmNtitle, "ISDAT cuiex"); n++;
    XtSetArg(args[n], XmNwidthInc, 300); n++;
    XtSetArg(args[n], XmNargc, argc); n++;
    XtSetArg(args[n], XmNargv, argv); n++;
    shellW = XtAppCreateShell(appName, "Cuiex",
                              applicationShellWidgetClass,
                              display, args, n);

    /* create rowcolumn widget */
    n = 0;
    XtSetArg(args[n], XmNspacing, 0); n++;
    XtSetArg(args[n], XmNmarginWidth, 0); n++;
    XtSetArg(args[n], XmNmarginHeight, 0); n++;
    rowcolW = XmCreateRowColumn(shellW, "row_col", args, n);
    XtManageChild(rowcolW);

    /* create menubar */
    n = 0;
    menubarW = XmCreateMenuBar(rowcolW, "menu_bar", args, n);
    XtManageChild(menubarW);

    /* create File cascade button in menubar */
    xmstr = XmStringCreateLtoR("File", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(args[n], XmNlabelString, xmstr); n++;
    fileCascadeW = XmCreateCascadeButton(menubarW, "file_cascade",
                                         args, n);
    XtManageChild(fileCascadeW);
    XmStringFree(xmstr);

    /* create File pulldown menu pane */
    n = 0;
    filePulldownW = XmCreatePulldownMenu(menubarW, "file_pulldown",
                                         args, n);

    /* add File pulldown id */
    n = 0;
    XtSetArg(args[n], XmNsubMenuId, filePulldownW); n++;
    XtSetValues(fileCascadeW, args, n);

    /* create Exit menu button */
    xmstr = XmStringCreateLtoR("Exit", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(args[n], XmNlabelString, xmstr); n++;
    exitButtonW = XmCreatePushButton(filePulldownW, "exit_Button",
                                     args, n );
    XtManageChild(exitButtonW);
    XmStringFree(xmstr);

    /* add Exit menu button callback */
    XtAddCallback (exitButtonW, XmNactivateCallback, Quit,NULL);

    /* create Parameter cascade button in menubar */
    n = 0;
    xmstr = XmStringCreateLtoR("Parameter", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(args[n], XmNlabelString, xmstr); n++;
    paramCascadeW = XmCreateCascadeButton(menubarW, "param_cascade",
                                          args, n);
    XtManageChild(paramCascadeW);
    XmStringFree (xmstr);

    /* create Parameter pulldown menu pane */
    n = 0;
    paramMenuW = XmCreatePulldownMenu(menubarW, "paramMenuW", args, n);

    /* add Parameter pulldown id */
    n = 0;
    XtSetArg(args[n], XmNsubMenuId, paramMenuW); n++;
    XtSetValues(paramCascadeW, args, n);

    /* create label */
    n = 0;
    xmstr = XmStringCreateLtoR("", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(args[n], XmNlabelString, xmstr); n++;
    XtSetArg(args[n], XmNmarginTop, 10); n++;
    XtSetArg(args[n], XmNmarginBottom, 10); n++;
    XtSetArg(args[n], XmNmarginLeft, 10); n++;
    XtSetArg(args[n], XmNmarginRight, 10); n++;
    labelW = XmCreateLabel(rowcolW, "label", args, n);
    XtManageChild(labelW);
    XmStringFree (xmstr);
}
```

B.4 File cb.c

```
/*
#MN Module name: $Id: cb.c,v 1.1 1995/03/08 18:24:34 al Exp al $
#MA Anders Lundgren, IRF-U

#MD Module description:
#MD cuiex callback functions.
*/


#include <Isstd.h>
#include <Is.h>
#include <Db.h> /* Data base declarations */
#include <Mo.h>
#include <Tb.h>
#include <Xm/Xm.h>
#include "gui.h"

static char *rcsId = "$Id: cb.c,v 1.1 1995/03/08 18:24:34 al Exp al $";

static IsTmInfo tmInfo; /* time manager info */

extern int specValid;
extern DbDataSpec dataSpec;


/*
#FN Function name: UpdataTimeCB
#FD Function description:
#FD Callback that is called when the time manager
#FD has sent us new time information.

#FID Interface:
#FI reason - integer code for callback reason
#FI data - pointer to the data
#FI closure - not used
*/
void UpdateTimeCB(reason, data, closure)
int reason; /* callback reason */
IsPointer data; /* orbit manager information */
IsPointer closure;
{
    IsTmInfo *info;

    info = (IsTmInfo *)data;

    /* save it */
    tmInfo = *info;

    /* update window */
    Update(info);
}

/*
#FN Function name: ParamCB
```

```
#FD Function description:
#FD Callback from the parameter menu.

#FID Interface:
#FI menuspec - menu specification handle
*/
void ParamCB(menuSpec)
TbMenuSpec *menuSpec;
{
    /* save it and record that it's valid */
    dataSpec = menuSpec->spec;
    specValid = 1;

    /* update display */
    Update(&tmInfo);
}

/*
#FN Function name: Quit
#FD Function description:
#FD Callback from the menu Exit button.
#FD Exits the client cuiex.
*/
void Quit(w, client_data, call_data )
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *call_data;
{
    exit(0);
}
```

B.5 File update.c

```
/*
#MN Module name: $Id: update.c,v 1.1 1995/03/08 18:24:34 al Exp al $
#MA Anders Lundgren, IRF-U

#MD Module description:
#MD Set of functions to get and display the data.
#MD The presented value will be the maximum value
#MD found during the requested interval.
*/

#include <Isstd.h>
#include <Is.h>
#include <Db.h> /* Data base declarations */
#include <Xm/Xm.h>
#include "gui.h"

static char *rcsId = "$Id: update.c,v 1.1 1995/03/08 18:24:34 al Exp al $";

static double GetMax();
static char *GetUnits();

DbDataObject *obj;
extern int specValid;
extern Database *db;
extern DbDataSpec dataSpec;


/*
#FN Function name: Update
#FD Function description:
#FD Updates and redisplays the text in the cuiex window.

#FID Interface:
#FI tmInfo - time manager information
*/
void Update(tmInfo)
IsTmInfo *tmInfo;
{
    int error;
    DbDataRequest req; /* data request */
    DbDataObject *obj; /* data object */
    char message[256];
    Arg args[20];

    /* make sure that user has selected a parameter */
    if (specValid) {
        /* describe the data we want */
        req.start = tmInfo->start;
        req.interval = tmInfo->interval;
        req.spec = dataSpec;
        req.units = DbUN_PHYS;
        req.reduction = DbRED_NONE;
        req.pack = DbPACK_TIMETAG;
        /* get the data from the database handler */
        error = DbGetData(db, &req, &obj);

        /* check if any errors */
        if (error == DbSUCCESS) {
            /* check if returned data is of right format */
            if (obj->rank == 0 && obj->dataType == DbTYPE_REAL_FLOAT &&
                obj->dimension == 0) {
                sprintf(message, "max: %g %s", GetMax(obj), GetUnits(obj));
             } else {
                sprintf(message,
                        "Can't handle\nrank = %d\ntype = %d\ndimension = %d",
                        obj->rank, obj->dataType, obj->dimension);
            }
            DbFreeDataObject(obj);
         } else {
            sprintf(message, "Error: %s", DbErrorString(error));
         }
    } else {
        strcpy(message, "No Parameter selected");
    }

    /* put message into window */
    XtSetArg(args[0], XmNlabelString,
             XmStringCreateLtoR(message, XmSTRING_DEFAULT_CHARSET));
    XtSetValues(labelW, args, 1);
}

/*
#FN Function name: GetMax
#FD Function description:
#FD Finds the maximum value of all samples in the data object.

#FID Interface:
#FI obj - pointer to data object
#FR Returns the maximum value as a double.
*/
static double GetMax(obj)
DbDataObject *obj;
{
    int i;
    int seg;
    double max;
    DbDataRF0 *rf0;

    /* set initial max value */
    rf0 = (DbDataRF0 *)obj->seg[0].data;
    max = rf0[0].re;

    /* loop over all samples looking for maximum value */
    for (seg = 0; seg < obj->segments; seg++) {
        rf0 = (DbDataRF0 *)obj->seg[seg].data;
        for (i = 0; i < obj->seg[seg].samples; i++) {
            if (rf0[i].re > max) max = rf0[i].re;
        }
    }
    return max;
}

/*
#FN Function name: GetUnits
#FD Function description:
#FD Gets the units of the data in the data object.

#FID Interface:
#FI obj - pointer to data object
#FR Returns the units as a string.
*/
static char *GetUnits(obj)
DbDataObject *obj;
{
    return obj->info[0].unitString;
}
```

B.6 File Imakefile

```
XCOMM $Id: Imakefile,v 1.1 1995/03/08 18:24:34 al Exp al $

/*
   Imake template for cuiex. Imake will generate a Makefile
   from this template.
   To regenerate the Makefile after a change to Imakefile type:
   make Makefile (or imkmf).
*/

        DEPLIBS = $(DEPTBLIB) $(DEPDBLIB) $(DEPISLIB) $(DEPISUTILLIB) \
                  $(DEPMOLIB)
LOCAL_LIBRARIES = $(TBLIB) $(DBLIB) $(ISLIB) $(ISUTILLIB) \
                  $(MOLIB) \
                  $(XMLIB) $(XTLIB) $(X11LIB) \
                  $(MATHLIB)
       INCLUDES = -I. $(XMINC) $(XTINC) $(X11INC)

           SRCS = gui.c main.c cb.c update.c
           OBJS = gui.o main.o cb.o update.o

ComplexProgramTarget(cuiex)
RegisterClient(cuiex,demo,cuiex)

/* The design is not created by XDesigner but the rule is used
   to install the resource file only */
XDesignerInstall(Cuiex)
```

C Dblib library calls

```
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.
```

D Islib library calls

```
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)
```

E Isutillib library calls

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)
```