ISDAT 1.0 Programmers Guide
2. DBH Instrument modules

Anders Lundgren and Gunnar Holmgren
Swedish Institute of Space Physics
Uppsala Division
S-755 91 Uppsala
Sweden

December 1, 1999

Document Status Sheet			
1. Document Title: **ISDAT 1.0, Instrument modules**			
2. Document Reference Number: **CWD-IPG-002**			
3. Issue	4. Revision	5. Date	6. Reason for Change
Draft	0	94 Feb 14	New document.

Contents

1 Introduction

1.1 Purpose of the document

The purpose of this document is to provide the information needed to write instrument module code to be implemented in the ISDAT system. The document is intended for the programmer. A general introduction to the ISDAT is given in [Ref. 1]. An overview of documents related to coding and general guidelines are given in [Ref. 2].

1.2 Scope of the software

The scope of the ISDAT DBH instrument module software is to provide all necessary instrument specific knowledge needed to deliver requested data from a specific instrument to the clients.

1.3 Definitions and acronyms

Acronym	Meaning
DBH	Data Base Handler
GKS	Graphics Kernel System
ISDAT	Interactive Science Data Analysis Tool
N/A	Not Applicable
PEX	PHIGS Extension to X11
PHIGS	Programmer's Hierarchical Graphics System
TBD	To Be defined
TBW	To Be Written

2 Instrument module functions

The instrument module should provide all required information related to the specific instrument. A full implementation might be very extensive. Normally, only a subset of all functions are implemented in the first version and additionsl functions are added gradually. The functions of the DBH instrument modules are basically[1]:

- Registrer services at DBH core part[2]
- Read instrument associated data[3]
- Select the requested data channel[4].
- Unpack the raw data
- Apply calibration functions
- Identify and handle data gaps.
- Supply conceptual instrument information[5].
- Deliver data to the requesting client.
- Provide instrument specific character strings
- Provide instrument specific error messages.

Optional functions are:

- Provide content of on line data catalogue[6].
- Provide search functions[7]

[1]In the footnotes we use expressions xxFunction where xx stands for an instrument-unique identifier. For example Freja4GetData, EFWGetData
[2]handled by the xxInit function
[3]handled by the xxGetData function
[4]handled by the xxGetData function
[5]handled by the xxQuery function
[6]handled by the xxGetContent function
[7]handled by the xxSearch function

3 Instrument module coding

This guided tour is based on an imaginary instrument implemented as a template module listed in Appendix A. TBW

4 Instrument module installation

TBW

5 Instrument module documentation

All implemented instrument specific features shall be documented in an instrument specific DBH module document. The content of such a document might have the following list of contents, where all sections may not be applicable to a specific instrument module:

```
Title page including author, version, revision, date
i  Change sheet
ii List of contents
1  Introduction
1.1 Purpose of the document
1.2 Scope of the software
1.3 Definitions, acronyms
1.4 References
1.5 Overview of the document
2   Standards
2.1 Design standards
2.2 Documentation standards
2.3 Naming conventions
2.4 Programming standards
2.5 Software development tools
3   Component design specifications
3.n   File identifier
      Functions covered in the file
3.n.m Function identifier
3.n.m.1 Function
3.n.m.2 Subordinates
3.n.m.3 Returns:
3.n.m.4 References
3.n.m.5 Resources
3.n.m.6 Processing
3.n.m.7 Data
4   Conceptual instrument descriptions
4.n Sensor identifier
4.n.1 Physical parameter
4.n.2 Processing
4.n.3 Software units
4.n.4 Calibration files
4.n.5 Desc structure
4.n.6 Hierarchical sub-structure
5 Data
5.1 Index file
5.n Data file identifier
Appendix A Source code listing
```

References

[1] G. Holmgren and A. Lundgren. ISDAT interactive scientific analysis tool. an introduction. Technical report, IRF-U, February 1994.

[2] A. Lundgren and G. Holmgren. ISDAT programmers guide. 1. overview and general guidelines. Technical report, IRF-U, February 1994.

A Source code

A.1 File README

```
Implements a test instrument with support for GetContent,
GetData and Query.
The details of the fixed data sets are wired into the
code to make it as simple as possible. The intent was
to give an example where all data decommutation
complexity was removed to present a source code where
most of the code is used to interface to the ISDAT database
handler.

The logical instrument hierarchy is:

    project: Test
    member: -
    instrument: wave
    sensor: sine, square
    signal: wf, fft
    channel: -
    parameter: -

The fft signal is only available when using the square sensor.

The sampling rate is 450 Hz and the length of a format is 1 second.

Two data sets are available:

    930801 151000 - 930801 151140
    930802 183000 - 930802 183140

The second data set has a data drop at 930802 183005 with a
duration of 3 seconds.
The sine and square waves have a frequency of 1 Hz.


FILES:

GetContent.c    GetContent entry point. Depends on:
CoInterval.c

GetData.c GetData entry point. Depends on:
Fft.c, Locate.c, Util.c, Wave.c

Imakefile Template to generate Makefile.

Init.c Init entry point. Registrates GetContent,
GetData and Query.

InitData.c Creates the data set. Called from Init.c.

Query.c Query entry point. Describes the logical
instrument hierarchy to the client.
```

```
Test.h Definition of logical instrument hierarchy values.

T.h Definition of some useful parameters.

Orbit.h Definition of data set structure and a state
structure (Orbit).
```

A.2 File Fft.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "datastr.h"
#include "T.h"

/*
 * TestFftData()
 *
 * Returns one format of fft data.
 *
 */
int TestFftData(pOrbit, desc, dest, time)
Orbit *pOrbit;
DataDescPtr desc;
float **dest;
IsTime *time;
{
    int n;
    int sec;
    float dropValue = DROP_VALUE;
    DataSet *dp;
    void TestTimeTagging();

    if (!dest) {
/* calculate the maximum number of samples it will take to cover
   the specified interval including some extra formats as margin */
n = IsTime2Float(&desc->interval) + FORMAT_MARGIN;
return n;
    }

    pOrbit->error = DbNOT_IMPLEMENTED;
    return 0;
}
```

A.3 File GetContent.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "commstr.h"
#include "datastr.h"
#include "T.h"

extern DataSet *testData;
```

```
/*
 * TestGetContent()
 *
 * GetContent entry.
 *
 */
int TestGetContent(client, desc, secp)
ClientPtr client;
ContentDescPtr desc;
ContentSection **secp;
{
    return AllContent(client, desc, secp);
}

/*
 * AllContent()
 *
 * Handles GetContent when mode is DbCONTENT_ALL.
 *
 */
static int AllContent(client, desc, secp)
ClientPtr client;
ContentDescPtr desc;
ContentSection **secp;
{
    ContentSection *p;

    *secp = (ContentSection *)malloc(2 * sizeof(ContentSection));
    p = *secp;

    /* first data set */
    /* set start time of online data and its interval */
    p->period.start = testData[0].start;
    p->period.interval = testData[0].interval;
    p->message[0] = '\0';
    p++;

    /* second data set */
    /* set start time of online data and its interval */
    p->period.start = testData[1].start;
    p->period.interval = testData[1].interval;
    p->message[0] = '\0';

    desc->sections = 2;
    return DbSUCCESS;
}
```

A.4 File GetData.c

```
#include <stdio.h>
#include <math.h>
#include "Dbdef.h"
#include "commstr.h"
#include "datastr.h"
```

```
#include "T.h"


/*
 * TestGetData()
 *
 * GetData entry point.
 *
 */
int TestGetData(client, desc, data, time)
ClientPtr client;
DataDescPtr desc;
float **data;
IsTime **time;
{
    int i;
    int samplesAllocated; /* samples we have allocated storage for */
    int samplesNow; /* current number of samples */
    int samplesSegment; /* current number of samples in segment */
    int samplesNeeded; /* number of samples within the interval */
    int samplesUnused; /* number of unused samples at the end */
    IsTime currentTime;
    int n; /* number of samples returned from data func */
    int lastn;
    int drop; /* set if a drop is detected */
    int dropStart; /* set if a drop is present at the start */
    int fftReq; /* set if fft request (2-dimensional data) */
    int segment; /* current segment index */
    int (*func)(); /* current data function */
    IsTime oldStw;
    float *pd; /* 1-dimensional data array pointer */
    float **pm; /* 2-dimensional data array pointer */
    IsTime *pt; /* time tag array pointer */
    IsTime stop; /* stop time of requested interval */
    IsTime tmp; /* scratch variable */
    IsTime end; /* scratch variable */
    IsTime tForm; /* duration of one format */
    double dsec; /* scratch variable */
    double diff; /* scratch variable */
    Orbit *pOrbit; /* state pointer */
    double dtFormat; /* duration of one format */
    Segment *segp; /* pointer to current segment */
    static Segment *segmentTable = 0;
    Orbit *TestLocateData();
    int (*TestGetDataFunc())();

    /* set some constant desc members */
    desc->sys.segmentType = SEG_CONTIGUOUS;
    /* tell upper layer to free storage for data and time arrays when done */
    desc->sys.freeData = 1;
    desc->sys.freeTime = 1;

    dropStart = 0;
    desc->warning = 0;
    fftReq = desc->spec.signal == DbTEST_FFT;
    desc->dimension = (fftReq) ? 2 : 1;
    /* try to find a data set that covers the requested time */
    pOrbit = TestLocateData(desc);
    if (pOrbit == NULL) {
return DbBAD_TIME;
    }
    pOrbit->error = DbSUCCESS;

    if (!segmentTable) {
/* allocate storage for a big enough segment table */
segmentTable = (Segment *)malloc(MAX_SEGMENTS * sizeof(Segment));
    }
    desc->sys.segmentPtr = segmentTable;
    segp = segmentTable;

    stop = desc->start;
    IsAddTime(&stop, &desc->interval);
    currentTime = desc->start;
    pOrbit->first = 1;

    func = TestGetDataFunc(desc, pOrbit);
    if (pOrbit->error != DbSUCCESS) return pOrbit->error;

    /* the first call to the data function is just a question
       of how many samples will be needed in the data array */
    samplesAllocated = (*func)(pOrbit, desc, NULL, NULL);

    if (fftReq) {
desc->sys.size = sizeof(float);
desc->sys.allocated[0] = samplesAllocated;
desc->sys.allocated[1] = 512;
*data = (float *)VaVmalloc(desc->sys.size, 2,
   desc->sys.allocated[0],
   desc->sys.allocated[1],
   &desc->sys.bytes);
pm = (float **)*data;
    } else {
desc->sys.size = sizeof(float);
desc->sys.allocated[0] = samplesAllocated;
*data = (float *)malloc((unsigned)(samplesAllocated * sizeof(float)));
pd = *data;
    }
    if (desc->sys.pack == DbPACK_TIMETAG) {
*time = (IsTime *)malloc((unsigned)(samplesAllocated * sizeof(IsTime)));
pt = *time;
    }
    samplesNow = 0;
    samplesSegment = 0;
    samplesUnused = 0;
    segment = 0;
    desc->sys.offset = 0;

    /* enter loop collecting data one format at a time */
    for (;;) {
if (fftReq) {
    n = (*TestGetDataFunc(desc, pOrbit))(pOrbit, desc, pm, pt);
```

```
} else {
    n = (*TestGetDataFunc(desc, pOrbit))(pOrbit, desc, pd, pt);
}
if (pOrbit->error == DbBAD_EOF) {
    pOrbit->error = DbSUCCESS;
    desc->warning |= DbWARN_EOF;
    return pOrbit->error;
}
if (pOrbit->error != DbSUCCESS) {
    return pOrbit->error;
}
dtFormat = pOrbit->t;
tForm = IsFloat2Time(dtFormat);

if (IsCmpTime(&pOrbit->istw, &desc->start) > 0)
    currentTime = pOrbit->istw;
if (pOrbit->first) {
    if (IsCmpTime(&stop, &pOrbit->istw) <= 0) {
/* got a format past the requested interval */
/* if no error code was set, it is set to a drop */
if (!pOrbit->error) pOrbit->error = DbBAD_DROP;
return pOrbit->error;
    }
    tmp = desc->start;
    IsSubTime(&tmp, &pOrbit->istw);
    diff = IsTime2Float(&tmp);
    if (diff < 0) dropStart = 1;
    /* calculate number of samples at the beginning of the
       first format to throw away */
    if (diff > 0) desc->sys.offset = (diff * n) / dtFormat + 1;
    oldStw = desc->start;
    segp->offset = 0;
    dsec = dtFormat * desc->sys.offset / n;
    segp->start = pOrbit->istw;
    IsAddTimeFloat(&segp->start, dsec);
}

end = oldStw;
IsAddTime(&end, &tForm);
tmp = pOrbit->istw;
IsSubTime(&tmp, &end);
diff = IsTime2Float(&tmp);
drop =  (diff > dtFormat / 2.0);
if (!pOrbit->first && drop) {
    /* fininsh up the current segment and start a new segment */
    segp->interval = end;
    IsSubTime(&segp->interval, &segp->start);
    if (!segment) samplesSegment -= desc->sys.offset;
    segp->samples = samplesSegment;
    segment++;
    segp++;
    segp->start = pOrbit->istw;
    segp->offset = samplesNow - desc->sys.offset;
    samplesSegment = 0;
}
if (dropStart || drop) {
    desc->warning |= DbWARN_DROP;
    pOrbit->error = DbSUCCESS;
}
if (IsCmpTime(&currentTime, &stop) > 0) break;

samplesNow += n;
samplesSegment += n;
if (fftReq) pm += n; else pd += n;
pt += n;
oldStw = pOrbit->istw;
lastn = n;
next_loop:
dropStart = 0;
tmp = pOrbit->istw;
IsAddTime(&tmp, &tForm);
IsSubTime(&tmp, &stop);
dsec = IsTime2Float(&tmp);
if (dsec > 0) {
    /* that's it, we've got all the requested data */
    samplesUnused = lastn * dsec / dtFormat;
    break;
}
pOrbit->first = 0;
    }

    if (IsCmpTime(&currentTime, &stop) <= 0) {
/* complete the current (last) segment */
end = currentTime;
IsAddTime(&end, &tForm);
dsec = dtFormat * samplesUnused / lastn;
IsSubTimeFloat(&end, dsec);
segp->interval = end;
IsSubTime(&segp->interval, &segp->start);
if (!segment) samplesSegment -= desc->sys.offset;
samplesSegment -= samplesUnused;
segp->samples = samplesSegment;
segment++;
segp++;
    }
    desc->sys.segments = segment;

    samplesNeeded = samplesNow - desc->sys.offset - samplesUnused;
    if (samplesNow && !samplesNeeded) {
return DbBAD_ZONE;
    }

    desc->samples[0] = samplesNeeded;
    if (!fftReq) {
desc->sys.reduce = 1;
desc->sys.bytes = desc->samples[0] * sizeof(float);
    }
    if (fftReq) {
desc->info.samplingFreq[0] = pOrbit->fftRate;
desc->info.samplingFreq[1] = pOrbit->rate;
    } else {
desc->info.samplingFreq[0] = pOrbit->rate;
```

```
    }

    return pOrbit->error;
}

/*
 * TestGetDataFunc()
 *
 * Returns the data function corresponding to the specified
 * logical instrument.
 *
 */
static int (*TestGetDataFunc(desc, pOrbit))()
DataDescPtr desc;
Orbit *pOrbit;
{
    int TestWaveData();
    int TestFftData();

    switch (desc->spec.signal) {
    case DbTEST_WF:
return TestWaveData;
break;
    case DbTEST_FFT:
return TestFftData;
break;
    }
    pOrbit->error = DbBAD_INSTRUMENT;
    return 0;
}
```

A.5 File Init.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "datastr.h"
#include "Test.h"


/*
 * TestInit()
 *
 * This function is automatically found by the ISDAT configuration
 * step and used to registrate functions supported by this
 * implementation. It is called once when the database handler
 * is started.
 * In this implementation it is used to create the two data sets as well.
 *
 */
void TestInit()
{
    DataSpec spec;
    extern int TestGetContent();
    extern int TestGetData();
    extern int TestQuery();
    extern int TestSearch();

    spec.project = DbTEST;
    spec.member = DbUNDEF;
    spec.instrument = DbTEST_WAVE;
    spec.sensor = DbUNDEF;
    spec.signal = DbUNDEF;
    spec.channel = DbUNDEF;
    spec.parameter = DbUNDEF;
    RegisterFunction(FUNC_GETCONTENT, &spec, TestGetContent);
    RegisterFunction(FUNC_GETDATA, &spec, TestGetData);
    RegisterFunction(FUNC_QUERY, &spec, TestQuery);
    RegisterFunction(FUNC_SEARCH, &spec, TestSearch);

    /* create data sets */
    TestInitializeData();
}
```

A.6 File InitData.c

```
#include <stdio.h>
#include <math.h>
#include "T.h"

DataSet *testData;

/*
 * TestInitializeData()
 *
 * Creates the two data sets.
 *
 */
TestInitializeData()
{
    int i;
    double squareValue = 1.0;

    testData = (DataSet *)malloc(2 * sizeof(DataSet));

    IsYmdHms2Time("930801 151000", &testData[0].start);
    testData[0].interval = IsFloat2Time(100.0);

    IsYmdHms2Time("930802 183000", &testData[1].start);
    testData[1].interval = IsFloat2Time(100.0);

    /* create sine and square data */
    for (i = 0; i < INTERVAL * SRATE; i++) {
testData[0].sine[i] = sin(2.0 * M_PI / (double)SRATE * i);
testData[0].square[i] = squareValue;
testData[1].sine[i] = cos(2.0 * M_PI / (double)SRATE * i);
testData[1].square[i] = -squareValue;
if (!(i % (SRATE / 2))) squareValue = -squareValue;
    }
    /* fake a 3 second drop after 5 seconds in the second data set */
    for (i = 5 * SRATE; i < 5 * SRATE + 3 * SRATE; i++) {
testData[1].sine[i] = DROP_VALUE;
testData[1].square[i] = DROP_VALUE;
    }

    /* koko! add FFT code ! */
}
```

A.7 File Locate.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "datastr.h"
#include "T.h"

static Orbit orb;
extern DataSet *testData;

/*
 * TestLocateData()
 *
 * Locates the data set covering the requested time.
 * Returns a null pointer if neither of the two data
 * sets is within the requested time.
 *
 */
Orbit *TestLocateData(desc)
DataDescPtr desc;
{
    int i;
    DataSet *p;
    IsTime stop;

    for (i = 0; i < 2; i++) {
p = testData + i;
stop = p->start;
IsAddTime(&stop, &p->interval);
if (IsCmpTime(&desc->start, &p->start) >= 0 &&
    IsCmpTime(&desc->start, &stop) < 0) {
    orb.data = p;
    return &orb;
}
    }

    return (Orbit *)0;
}
```

A.8 File Query.c

```
#include "Dbdef.h"
#include "Test.h"
#include "commstr.h"
#include "datastr.h"
```

```
/*
 * TestQuery()
 *
 * Query entry point.
 *
 */
int TestQuery(client, desc, qdata)
ClientPtr client;
QueryDescPtr desc;
QueryData *qdata;
{
    int n;

    switch (desc->level) {
    case DbLEVEL_PROJECT:
n = QueryProject(desc->mode, qdata);
break;
    case DbLEVEL_MEMBER:
n = QueryMember(desc->mode, desc->spec.project, qdata);
break;
    case DbLEVEL_INSTRUMENT:
n = QueryInstrument(desc->mode, desc->spec.project,
    desc->spec.member, qdata);
break;
    case DbLEVEL_SENSOR:
n = QuerySensor(desc->mode, desc->spec.project, desc->spec.member,
desc->spec.instrument, qdata);
break;
    case DbLEVEL_SIGNAL:
n = QuerySignal(desc->mode, desc->spec.project,
desc->spec.member, desc->spec.instrument,
desc->spec.sensor, qdata);
break;
    case DbLEVEL_CHANNEL:
n = QueryChannel(desc->mode, desc->spec.project,
 desc->spec.member, desc->spec.instrument,
 desc->spec.sensor, desc->spec.signal, qdata);
break;
    case DbLEVEL_PARAMETER:
n = QueryParameter(desc->mode, desc->spec.project,
   desc->spec.member, desc->spec.instrument,
   desc->spec.sensor, desc->spec.signal,
   desc->spec.channel, qdata);
break;
    }
    return n;
}

static QueryProject(mode, p)
int mode;
QueryData *p;
{
    int n = 0;
    p[n].value = DbTEST;
    p[n++].name = "Test";
    return n;
}

static QueryMember(mode, project, p)
int mode;
int project;
QueryData *p;
{
    int n = 0;

    switch (project) {
    case DbTEST:
p[n].value = DbUNUSED;
p[n++].name = "*";
break;
    }
    return n;
}

static QueryInstrument(mode, project, member, p)
int mode;
int project;
int member;
QueryData *p;
{
    int n = 0;

    switch (project) {
    case DbTEST:
p[n].value = DbTEST_WAVE;
p[n++].name = "wave";
    }
    return n;
}

static QuerySensor(mode, project, member, instrument, p)
int mode;
int project;
int member;
int instrument;
QueryData *p;
{
    int n = 0;

    switch (project) {
    case DbTEST:
switch (instrument) {
case DbTEST_WAVE:
    p[n].value = DbTEST_SINE;
    p[n++].name = "sine";
    p[n].value = DbTEST_SQUARE;
    p[n++].name = "square";
    break;
}
break;
    }
    return n;
}

static QuerySignal(mode, project, member, instrument, sensor, p)
int mode;
int project;
int member;
int instrument;
int sensor;
QueryData *p;
{
    int n = 0;

    switch (project) {
    case DbTEST:
        switch (instrument) {
        case DbTEST_WAVE:
    switch (sensor) {
    case DbTEST_SINE:
p[n].value = DbTEST_WF;
p[n++].name = "wf";
break;
    case DbTEST_SQUARE:
p[n].value = DbTEST_WF;
p[n++].name = "wf";
p[n].value = DbTEST_FFT;
p[n++].name = "fft";
break;
    }
    break;
}
break;
    }
    return n;
}

static QueryChannel(mode, project, member, instrument, sensor, signal, p)
int mode;
int project;
int member;
int instrument;
int sensor;
QueryData *p;
{
    int n = 0;

    return n;
}

static QueryParameter(mode, project, member, instrument, sensor, signal,
                      channel, p)
int mode;
int project;
int member;
int instrument;
int sensor;
int channel;
QueryData *p;
{
    int n = 0;

    return n;
}
```

A.9 File Search.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "commstr.h"
#include "datastr.h"
#include "T.h"

extern DataSet *testData;


/*
 * TestSearch()
 *
 * Handles Search.
 *
 */
int TestSearch(client, desc, secp)
ClientPtr client;
SearchDescPtr desc;
SearchSection **secp;
{
    int n;
    SearchSection *sp;
    DataSpec spec;

    desc->sections = 0;

    *secp = (SearchSection *)malloc(3 * sizeof(SearchSection));
    sp = *secp;
    sp[0].items = 0;
    sp[1].items = 0;
    sp[2].items = 0;

    memset(&spec, 0, sizeof(DataSpec));

    /* add Test->wave->sine->wf section if requested and within interval */
    spec.project = DbTEST;
    spec.instrument = DbTEST_WAVE;
    spec.sensor = DbTEST_SINE;
    spec.signal = DbTEST_WF;
    if (!SpecCompare3(&desc->spec, &spec)) {
n = AddIt(desc, sp, &spec);
if (n) {
    desc->sections++;
    sp++;
}
    }

    /* add Test->wave->square->wf section if requested and within interval */
    spec.project = DbTEST;
    spec.instrument = DbTEST_WAVE;
    spec.sensor = DbTEST_SQUARE;
    spec.signal = DbTEST_WF;
    if (!SpecCompare3(&desc->spec, &spec)) {
n = AddIt(desc, sp, &spec);
if (n) {
    desc->sections++;
    sp++;
}
    }

    /* add Test->wave->square->fft section if requested and within interval */
    spec.project = DbTEST;
    spec.instrument = DbTEST_WAVE;
    spec.sensor = DbTEST_SQUARE;
    spec.signal = DbTEST_FFT;
    if (!SpecCompare3(&desc->spec, &spec)) {
n = AddIt(desc, sp, &spec);
if (n) {
    desc->sections++;
    sp++;
}
    }

    if (desc->sections == 0) {
/* free storage if no sections were added */
free(*secp);
*secp = (SearchSection *)0;
    }

    return DbSUCCESS;
}

/*
 * AddIt()
 *
 * Adds a new section for spec covering all data sets
 * within the specified interval.
 * Returns zero if the section was not added.
 *
 */
static int AddIt(desc, sp, spec)
SearchDescPtr desc;
SearchSection *sp;
DataSpec *spec;
{
    int n = 0;
    IsTime start;
    IsTime stop;
    /* process 930801 151000 - 930801 151140 */
    start = testData[0].start;
    stop = testData[0].start;
    IsAddTime(&stop, &testData[0].interval);
    n += CheckInterval(desc, sp, spec, &start, &stop);

    /* process 930802 183000 - 930802 183005 */
    start = testData[1].start;
    stop = testData[1].start;
    IsAddTimeFloat(&stop, 5.0);
    n += CheckInterval(desc, sp, spec, &start, &stop);

    /* process 930802 183008 - 930802 183140 */
    start = testData[1].start;
    IsAddTimeFloat(&start, 5.0 + 3.0);
    stop = testData[1].start;
    IsAddTime(&stop, &testData[1].interval);
    n += CheckInterval(desc, sp, spec, &start, &stop);

    return n;
}

/*
 * CheckInterval()
 *
 * Checks if the requested interval is covered by start/stop
 * and if it is the period is added to the section.
 * Returns one if a period was added to the content list.
 *
 */
static int CheckInterval(desc, sp, spec, start, stop)
SearchDescPtr desc;
SearchSection *sp;
DataSpec *spec;
IsTime *start;
IsTime *stop;
{
    int coverage = 0;
    IsTime reqStop;
    IsTime interval;

    reqStop = desc->start;
    IsAddTime(&reqStop, &desc->interval);

    /* check if requested start is inside the interval */
    if (IsCmpTime(stop, &desc->start) > 0 &&
IsCmpTime(&desc->start, start) > 0) coverage++;

    /* check if requested stop is inside the interval */
    if (IsCmpTime(stop, &reqStop) > 0 &&
IsCmpTime(&reqStop, start) > 0) coverage++;

    /* check if requested start/stop surrounds the interval */
    if (IsCmpTime(start, &desc->start) > 0 &&
IsCmpTime(&reqStop, stop) > 0) coverage++;
```

```
    if (!coverage) return 0;

    /* update the section and allocate storage for the content
       if this is the first period */
    if (!sp->items) sp->period = (TimePeriod *)malloc(3 * sizeof(TimePeriod));
    interval = *stop;
    IsSubTime(&interval, start);
    sp->spec = *spec;
    sp->period[sp->items].start = *start;
    sp->period[sp->items].interval = interval;
    sp->message[0] = '\0';
    sp->items++;

    return 1;
}

/*
 * SpecCompare3()
 *
 * Checks that the specifications a and b matches.
 * An element may be zero to indicate a wildcard
 * to match anything.
 * Only sensors and signals are compared.
 * Returns zero if equal.
 *
 */
#define COMPARE(x) \
    if (a->x && b->x && a->x != b->x) return 1;

static int SpecCompare3(a, b)
DataSpec *a;
DataSpec *b;
{
    COMPARE(sensor)
    COMPARE(signal)
    return 0;
}
```

A.10 File Util.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "T.h"


/*
 * TestTimeTagging()
 *
 * Fills a time array with time values, one for each sample.
 *
 */
void TestTimeTagging(pOrbit, n, stime, time, dtForm)
Orbit *pOrbit;
int n; /* number of samples */
```

```
IsTime *stime; /* start time */
IsTime *time; /* time array */
double dtForm; /* number of seconds covered by this sample set */
{
    int i;
    double delta;
    IsTime current;

    delta = dtForm / n;
    current = *stime;
    for (i = 0; i < n; i++) {
*time = current;
IsAddTimeFloat(&current, delta);
time++;
    }
}
```

A.11 File Wave.c

```
#include <stdio.h>
#include "Dbdef.h"
#include "datastr.h"
#include "T.h"


/*
 * TestWaveData()
 *
 * Returns one format of sine or square data.
 *
 */
int TestWaveData(pOrbit, desc, dest, time)
Orbit *pOrbit;
DataDescPtr desc;
float *dest;
IsTime *time;
{
    int n;
    int sec;
    float dropValue = DROP_VALUE;
    float *wave;
    DataSet *dp;
    IsTime tmp;
    void TestTimeTagging();

    if (!dest) {
/* calculate the maximum number of samples it will take to cover
   the specified interval including some extra formats as margin */
n = 450.0 * IsTime2Float(&desc->interval) +
    450.0 * FORMAT_MARGIN;
return n;
    }

    dp = pOrbit->data;
    /* get correct waveform */
    wave = (desc->spec.sensor == DbTEST_SINE) ? dp->sine : dp->square;

    if (pOrbit->first) {
/* position at first format containing the requested sample,
   note that 1 format is 1 second */
tmp = desc->start;
IsSubTime(&tmp, &dp->start);
pOrbit->format = IsTime2Float(&tmp);
if (pOrbit->format >= INTERVAL) {
    pOrbit->error = DbBAD_TIME;
    return 0;
}
    }

    /* skip over drops */
    while (wave[pOrbit->format * SRATE] == dropValue &&
pOrbit->format < INTERVAL) {
pOrbit->format++;
    }

    if (pOrbit->format >= INTERVAL) {
pOrbit->error = DbBAD_EOF;
return 0;
    }

    n = SRATE;
    pOrbit->rate = SRATE;
    pOrbit->t = 1.0;
    /* calculate start time of this format */
    pOrbit->istw = dp->start;
    IsAddTimeFloat(&pOrbit->istw, (double)pOrbit->format);

    /* copy this formats data to the data array */
    memcpy(dest, wave + pOrbit->format * SRATE, n * sizeof(float));

    desc->units[0] = DbUN_VOLT;
    if (desc->sys.pack == DbPACK_TIMETAG) {
/* time tag if requested */
TestTimeTagging(pOrbit, n, &pOrbit->istw, time, 1.0);
    }

    /* increment format number for next time called */
    pOrbit->format++;

    return n;
}
```

A.12 File T.h

```
typedef struct _DataSet {
    IsTime start; /* data set start time */
    IsTime interval; /* length of data set */
    float sine[INTERVAL * SRATE]; /* sine data */
    float square[INTERVAL * SRATE]; /* square data */
    float fft[INTERVAL][512]; /* fft data */
} DataSet;

typedef struct _Orbit {
    DataSet *data; /* current data set */
    int format; /* current format number */
    IsTime istw; /* time of data packet returned by
   data packup function */
    int first; /* set to one the first time a data
   packup function is called */
    double rate; /* sampling rate */
    double fftRate; /* number of FFTs per second */
    double t; /* time covered by one format */
    int error; /* error flag */
} Orbit;
#include <Isutil.h>
#include "Test.h"

#define INTERVAL 100 /* 100 seconds of data */
#define SRATE 450 /* 450 Hz sampling rate */

#define DROP_VALUE 1.0e30 /* magic number to mark drops */

#define MAX_SEGMENTS 10 /* max number of segments in an interval */
#define FORMAT_MARGIN 3 /* allocate storage for 3 extra formats
   as margin */

#include "Orbit.h"
#ifndef TEST_H
#define TEST_H

/* member */

/* instrument */
#define DbTEST_WAVE 1

/* sensor */
#define DbTEST_SINE 1
#define DbTEST_SQUARE 2

/* signal */
#define DbTEST_WF 1
#define DbTEST_FFT 2

/* channel */

#endif /* TEST_H */
#include <Server.tmpl>

#if HasMmap
   MMAP_DEFINES = -DHAS_MMAP
#endif
```

A.13 File Imakefile

```
SRCS =\
Fft.c \
GetContent.c \
GetData.c \
Init.c \
InitData.c \
Locate.c \
Query.c \
Search.c \
Util.c \
Wave.c

OBJS =\
Fft.o \
GetContent.o \
GetData.o \
Init.o \
InitData.o \
Locate.o \
Query.o \
Search.o \
Util.o \
Wave.o

    INCLUDES = -I. -I../include -I$(DINCLUDESRC) -I$(INCLUDESRC)
     DEFINES = $(MMAP_DEFINES)
    LINTLIBS = ../ddx/snf/llib-lsnf.ln ../os/4.2bsd/llib-los.ln

CppFortranObjectRule()
NormalLibraryObjectRule()
NormalLibraryTarget(Test,$(OBJS))
LintLibraryTarget(Test,$(SRCS))
NormalLintTarget($(SRCS))
InstrumentInclude(Test.h)

DependTarget()
```