Automated Software Configuration in the MONSOON System

P. N. Daly, N. C. Buchholz and P. Moore

MONSOON Project, Major Instrumentation Program, National Optical Astronomy Observatory, 950 N. Cherry Avenue, P. O. Box 26732, Tucson, AZ 85726-6732, USA.

ABSTRACT

MONSOON is the next generation OUV-IR controller project being developed at NOAO. The design is flexible, emphasizing code re-use, maintainability and scalability as key factors. The software needs to support widely divergent detector systems ranging from multi-chip mosaics (for LSST, QUOTA, ODI and NEWFIRM) down to large single or multi-detector laboratory development systems. In order for this flexibility to be effective and safe, the software must be able to configure itself to the requirements of the attached detector system at startup. The basic building block of all MONSOON systems is the PAN-DHE pair which make up a single data acquisition node. In this paper we discuss the software solutions used in the automatic PAN configuration system.

Keywords: OUV-IR controllers, MONSOON, GPX, Software Configuration

1. INTRODUCTION

The MONSOON image acquisition system is being developed to handle NOAO's present and future requirements for serving focal plane data to higher level software. In old parlance, it is an array controller. In the brave, new world, it is designed to handle both optical and infra-red data seamlessly—independent of science acquisition type. That is, when fully developed, MONSOON may be applied to single chip CCD or IR detectors, large area mosaics, photometric imaging, spectroscopy and technical imaging (the latter, for example, comprises fast guiding and wavefront sensing).

Such a wide variety of uses demands scalability via replication of software and hardware plus the ability to handle specific detectors system quirks and personalities. It must also be amenable to use by diverse user groups such as hardware engineers, software engineers, technical support staff and science users.

Figure 1 shows a schematic of the design of MONSOON as applied to the NOAO Extremely Wide-Field Infra-Red Mosaic (NEWFIRM) camera.[1,2] In this application, a single supervisor node controls two PAN-DHE pairs with data delivered to the data handling system (DHS) without flowing through the observation control system (OCS) first.

2. MULTI-LEVEL CONFIGURATION AND OPERATION

Normally, there is only one point of entry into the system via the GPX dictionary.[3–5] There is, however, a close mapping of GPX commands with PPX commands as implemented within the PAN. Therefore, a compound statement such as:

```
gpxSetAVP fSamples=16 intTime=3.6 coadds=2 pxlRows=4096 pxlCols=4096
```

may be broken down into a sequence of statements such as:

Further author information: (Send correspondence to PND)
PND: pnd@noao.edu, NCB: ncb@noao.edu, PM: pmoore@noao.edu

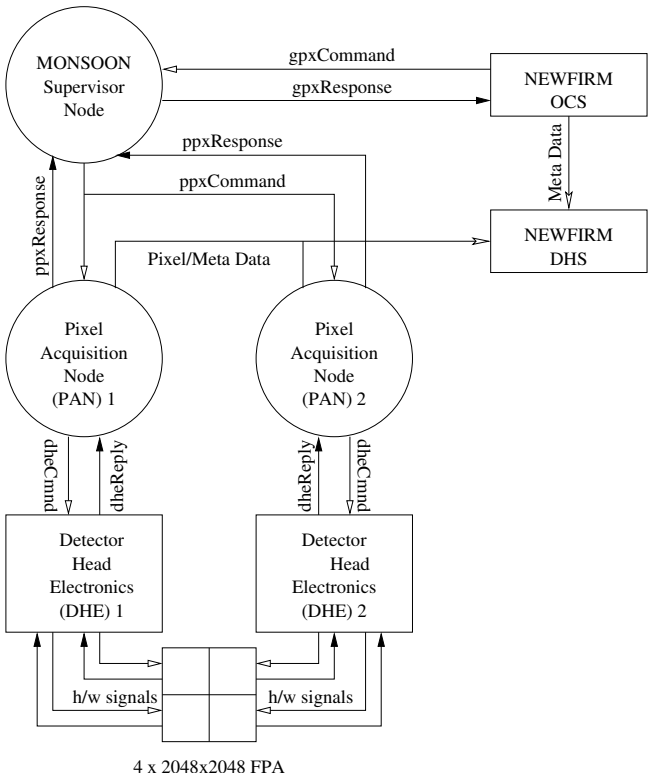


Figure 1. Schematic of a MONSOON System for NEWFIRM.

```
ppxSetAVP fSamples=16
ppxSetAVP intTime=3.6
ppxSetAVP coadds=2
ppxSetAVP pxlRows=4096
ppxSetAVP pxlCols=4096
```

To absolve the astronomer from typing in cumbersome command strings, the most common configurations are encapsulated within a named mode. The above command snippet, for example, could be part of a mode called, say, *FastIR* and the science client issues:

```
gpxSetMode FastIR
```

This goes, too, for higher-level software and/or GUI tools.

For software engineers, access to the system is via the GPX interface *or* the PPX interface. Hardware engineers have similar access but can also download code into the DHEs FPGA. Laboratory testing, however, would still be cumbersome if we required the technical engineers to type commands such as those shown in the above examples. Also, to handle the unique qualities of differing chips and technologies, we require a sub-set of commands unique to the system being developed or operated. We also require that such commands have names that are meaningful to the engineers working closely on the details of the system.

What is needed is a flexible and extensible way of defining commands for engineering access and persistence of configuration files in the system.

3. THE CONFIGURATION FILE

As with all such systems, there is a unique configuration file that can be loaded at run-time. In our system, this file is an ASCII, comma-separated values (CSV) file with the fields defined in Table 1. Comment lines begin with '#' and are ignored. So, a typical entry might look like this:

```
intTime,MCB_SEQITR,0x10000,1,0x2000000,SIMPLE,SIMPLE,FLOAT,ULONG,1000,0,LINEAR,0.0,65536.0,s,Integration time to 1 ms.
```

Table 1. MONSOON Configuration File CSV Fields.

Field Name	SQL Data Type	Description	Example (from above)
panName	varchar(32)	Command name as known to PAN	intTime
dheName	varchar(32)	Command name as known to DHE	MCB_SEQITR
baseAddr	integer	Base address in DHE for value	0x10000
nelms	integer	Number of elements in array	1
creg	integer	Miscellaneous control register	0x2000000
setMethod	text	Function to call to encode a value	SIMPLE
getMethod	text	Function to call to decode a value	SIMPLE
panType	text	Data type as known to the PAN	FLOAT
dheSize	text	Data size as known to the DHE	ULONG
coef1	float8	Coefficient 1 for conversion function	1000.0
coef2	float8	Coefficient 2 for conversion function	0.0
funcID	text	Conversion function	LINEAR
minValue	float8	Minimum value (before conversion)	0.0
maxValue	float8	Maximum value (before conversion)	65536.0
units	varchar(32)	Units (if applicable)	s
help	varchar(128)	Associated help text	Integration time to 1 ms.

Some points to note are:

panName is the name as known to the PAN and *must* be unique.

dheName is the name as known to the PAN and *need not* be unique.

baseAddr is a hexadecimal address. If 0x0, it is an ignored field.

nelms is the number of elements. If 0 it is ignored, if >1 an array is created.

creg is a control register for save/restore and GUI display flags.

setMethod/getMethod can only be one of the pre-existing methods: NOMETHOD, SIMPLE, INTTIME, ROISETC, ROISETR, FNAMESET, SETVOFF, SETBIASV, SETFBIAS, WRT2READ, STRINGSET, RDMSKWRT. Further methods can be added as required for future expansion.

panType can only be one of CHAR, UCHAR, SHORT, USHORT, INT, UINT, LONG, ULONG, STRING, FLOAT.

dheSize can only be one of ONEBIT, BYTE, CHAR, UCHAR, LONG, ULONG, SHORT, USHORT, TWLVBIT, TWNT4BIT.

funcID can only be one of the pre-existing functions: ERROR, LINEAR, POWER, TABLE1, BITFIELD, NONE. Further functions can be added as required for future expansion.

3.1. Loading the Command Tables

The MONSOON PAN software maintains two linked-list tables named PPX and ATT. The former, the PPX table, contains the well-known PPX commands (mapped from the well-defined GPX dictionary) and various command and control structures. The latter, the ATT table, is of variable length and is configured from the commands defined in the CSV configuration file. That is, upon initialization, the PAN software reads the CSV configuration file, parses it and then configures the data into the ATT table*:

```
/* initialize the command configuration tables */
cfgInit(&istat,resp,(cmdSMemE **)&cmdPtr);
if ( STATUS_BAD(istat) ) MNSN_REPORT(stderr,resp);

/* now populate the built-in command table into the PPX command list */
ppxInit(&istat,resp,(cmdTblS)&(cmdPtr->cmdList_PPX));
if ( STATUS_BAD(istat) ) MNSN_REPORT(stderr,resp);

/* now start the ATT commands loading by opening the CSV configuration file */
cfgOpenFile(&istat,resp,&fd);
if ( STATUS_BAD(istat) ) MNSN_REPORT(stderr,resp);

/* while there are no errors, populate the ATT command table */
while ( all_OK ) {

  /* read a line from the configuration file */
  (void) memset((void *)inline,0,sizeof(inline));
  all_OK = ( (fgets(inline,MAXLINE,fd)==(char *)NULL) ? FALSE : TRUE );

  /* if we have a valid line, parse it and add it into the structure */
  if ( all_OK && inline[0]!='#' && inline[0]!=',' && strlen(inline)>(size_t)1 ) {

    cmdCount++;
    cmdArgc = CLI_MAXLINE;
    cliParse(&istat,resp,CFG_DELIM_STR,inline,strlen(inline),&cmdArgc,cmdArgv);
    if ( STATUS_BAD(istat) ) MNSN_REPORT(stderr,resp);

    if ( cmdFind(&istat,(char *)NULL,cmdPtr->cmdList_ATT,cmdArgv[0]) == (cmdTblP)NULL ) {
      cmdCfgAdd(&istat,resp,&(cmdPtr->cmdList_ATT),FALSE,cmdArgc,cmdArgv);
      if ( STATUS_BAD(istat) ) MNSN_REPORT(stderr,resp);
    }
  }
}

/* close the configuration file */
cfgCloseFile(&istat,resp,fd);
if ( STATUS_BAD(istat) ) MNSN_REPORT(stderr,resp);
```

*The MONSOON software implements the notion of inherited status. If the status on entry is bad, the function returns immediately allowing us to defer error checking to a later stage whilst still retaining the place of the original error.

3.2. Command Execution

When a command is received by the PAN, it first searches the PPX table and, if found, executes the associated command/function. If the command is not found in the PPX table, it continues to search in the ATT table. At this point, if the command is recognized, a simple protocol is involved:

- If the ATT command has *no* associated parameter, it is executed as a read;
- If the ATT command has an associated parameter, it is executed as a write.

Thus, we can manipulate the integration time (defined by the CSV attribute *intTime*), for example, in a number of ways:

```
panDaemon> ppxSetAVP intTime=4.0
      DBG panControl: read "ppxSetAVP intTime=4.0"
      DBG panCommand: command found in PPX table at vector 0x4deec2b4
      DBG _cnvrttoDHE: generic: fValue = <4.000000>
      DBG cnvrttoDHE: generic: result = <4000>
      OK: intTime=4.0 - NEED EXACT SYNTAX

panDaemon> ppxGetAVP intTime
      DBG panControl: read "ppxGetAVP intTime=4.0"
      DBG panCommand: command found in PPX table at vector 0x4deeb814
      DBG getSimple attValue=0x00000fa0

panDaemon> intTime
      DBG panControl: read "intTime"
      DBG panCommand: command found in ATT table at vector 0x4deed3f8
      DBG getSimple attValue=0x00000fa0

panDaemon> intTime 6.6
      DBG panControl: read "intTime 6.6"
      DBG panCommand: command found in ATT table at vector 0x4deed3f8
      DBG _cnvrttoDHE: generic: fValue = <6.600000>
      DBG cnvrttoDHE: generic: result = <6599>

panDaemon> intTime
      DBG panControl: read "intTime"
      DBG panCommand: command found in ATT table at vector 0x4deed3f8
      DBG getSimple attValue=0x000019c7
```

3.3. Handling Array Attributes

If the *nelms* field is >1, an array is automatically created in the ATT table. Thus, for example, the configuration line:

```
mcbSeqLoopRegs,MCB_SEQLOOPREG,0x10110,16,0x8000000,SIMPLE,SIMPLE,FLOAT,ULONG,1,0,LINEAR,1,65535,Counts,loop registers
```

creates ATT entries *mcbSeqLoopRegs*[0] ... *mcbSeqLoopRegs*[15]. Thus, an individual element may be set:

```
panDaemon> mcbSeqLoopRegs[12] 44
```

or a *range* may be set in a single operation. For example, to set all elements between *mcbSeqLoopRegs*[2] and *mcbSeqLoopRegs*[8], we execute[†]:

[†]Strictly, speaking, at the time of writing this has yet to be implemented.

```
panDaemon> mcbSeqLoopRegs[] <2:8> 44
```

or *all* array elements may be set in a single operation:

```
panDaemon> mcbSeqLoopRegs[] 44
```

4. EVERYTHING BUT THE KITCHEN SINK

From the above, it is clear that adding a new command into the system is simply a matter of generating a new line in the CSV file with appropriate parameter values to enact the desired functionality. This elegant solution provides great flexibility to hardware and software engineering staff but it is not without its drawbacks:

- the Unix file clobbering mechanism can really ruin your day;
- there is no error checking on the CSV file with the potential that a typing error can fatally affect the hardware;
- there is no template to start from.

We have solved these problem by using a *PostGreSQL* database and creating a table with value checking where appropriate.

4.1. SQL

First we create the database and define wrapper functions to the MONSOON Star Date time stamping system[5]:

```
CREATE DATABASE monsoondb;
\c monsoondb
CREATE FUNCTION msdsql_utc( ) RETURNS int4   AS '/MNSN/soft_dev/lib/libmsdSql' LANGUAGE C;
CREATE FUNCTION msdsql_msd( ) RETURNS float8 AS '/MNSN/soft_dev/lib/libmsdSql' LANGUAGE C;
CREATE FUNCTION msdsql_mjd( ) RETURNS float8 AS '/MNSN/soft_dev/lib/libmsdSql' LANGUAGE C;
CREATE FUNCTION msdsql_jd( )  RETURNS float8 AS '/MNSN/soft_dev/lib/libmsdSql' LANGUAGE C;
CREATE FUNCTION msdsql_ld( )  RETURNS float8 AS '/MNSN/soft_dev/lib/libmsdSql' LANGUAGE C;
```

Then, we proceed to create a data table for each detector type. Here, we create a generic_I detector table:

```
CREATE TABLE generic_I (
  panName varchar(32),
  dheName varchar(32),
  baseAddr integer
    CHECK(baseAddr>=0),
  nelms integer
    CHECK(nelms>=0),
  creg integer
    CHECK(creg>=0),
  setMethod text
    CHECK(setMethod='NOMETHOD' OR setMethod='SIMPLE'    OR setMethod='INTTIME'  OR
          setMethod='ROISETC'  OR setMethod='ROISETR'   OR setMethod='FNAMESET' OR
          setMethod='SETVOFF'  OR setMethod='SETBIASV'  OR setMethod='SETFBIAS' OR
          setMethod='WRT2READ' OR setMethod='STRINGSET' OR setMethod='RDMSKWRT'),
  getMethod text
    CHECK(getMethod='NOMETHOD' OR getMethod='SIMPLE'    OR getMethod='INTTIME'  OR
          getMethod='ROISETC'  OR getMethod='ROISETR'   OR getMethod='FNAMESET' OR
```

```
        getMethod='SETVOFF'  OR getMethod='SETBIASV'  OR getMethod='SETFBIAS' OR
        getMethod='WRT2READ' OR getMethod='STRINGSET' OR getMethod='RDMSKWRT'),
  panType text
    CHECK(panType='CHAR'   OR panType='UCHAR' OR panType='SHORT' OR panType='USHORT' OR
        panType='INT'    OR panType='UINT'  OR panType='LONG'  OR panType='ULONG'  OR
        panType='STRING' OR panType='FLOAT'),
  dheSize text
    CHECK(dheSize='ONEBIT'  OR dheSize='BYTE'  OR dheSize='CHAR'  OR dheSize='UCHAR'  OR
        dheSize='LONG'    OR dheSize='ULONG' OR dheSize='SHORT' OR dheSize='USHORT' OR
        dheSize='TWLVBIT' OR dheSize='TWNT4BIT'),
  coef1 float8,
  coef2 float8,
  funcID text
    CHECK(funcID='ERROR'    OR funcID='LINEAR'   OR funcID='POWER' OR funcID='TABLE1' OR
        funcID='BITFIELD' OR funcID='NONE'),
  minValue float8,
  maxValue float8,
  units varchar(32),
  help varchar(128),
  id integer CHECK(id>=0),
  msd float8 CHECK(msd>0.0)
);
```

4.2. Perl

To populate this table, we use the present CSV file and run it through a Perl-script:

```
% csv2sql $MONSOON_CFG/generic_I.csv $MONSOON_CFG/generic_I.sql
```

Then, we can simply load the file within *psql* using the familiar '\i generic_I.sql' construct.

Clearly, we would like to manipulate the configuration records in a variety of ways. These include recovering rows from the appropriate database table and re-populating the database from other possible sources. Several Perl-scripts have been written, therefore, to facilitate the exchange of configuration data between the various formats. Eventually, the PAN software will read the database directly but, until then, we utilize the conversion utilities described in Table 2 on page 7.

Table 2. MONSOON Conversion Utility Routines.

Utility	Description	Usage
csv2sql	Converts CSV into SQL	csv2sql [in_file] [out_file]
cvs2xml	Converts CSV into XML	csv2xml [in_file] [out_file]
xml2csv	Converts XML into CSV	xml2csv [in_file] [out_file]
xml2sql	Converts XML into SQL	xml2sql [in_file] [out_file]
sql2csv	Recovers database records to CSV	sql2csv [db_name] [db_table] [out_file]
sql2xml	Recovers database records to XML	sql2xml [db_name] [db_table] [out_file]

4.3. XML, XSD and XSL

To display such data via a Web-browser interface requires using XML and a MONSOON schema. We have developed such a schema and style sheet for MONSOON configuration records. Whilst the schema source is too long to include in a paper of this length, the resulting XML output (for just the *intTime* record mentioned above) is shown below and the Web-browser view is shown in Figure 2 on page 9.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="generic_I.xsl"?>
<monsoonConfig detector="generic_I"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="monsoon.xsd">
 <monsoonRecord id="1">
  <panName>intTime</panName>
  <dheName>MCB_SEQITR</dheName>
  <baseAddr>0x00010000</baseAddr>
  <nelms>1</nelms>
  <creg>0x02000000</creg>
  <setMethod>SIMPLE</setMethod>
  <getMethod>SIMPLE</getMethod>
  <panType>FLOAT</panType>
  <dheSize>ULONG</dheSize>
  <coef1>1000.0</coef1>
  <coef2>0.0</coef2>
  <funcID>LINEAR</funcID>
  <minValue>0.0</minValue>
  <maxValue>65536.0</maxValue>
  <units>s</units>
  <help>Integration time to 1 ms.</help>
 </monsoonRecord>
</monsoonConfig>
```

4.4. Java

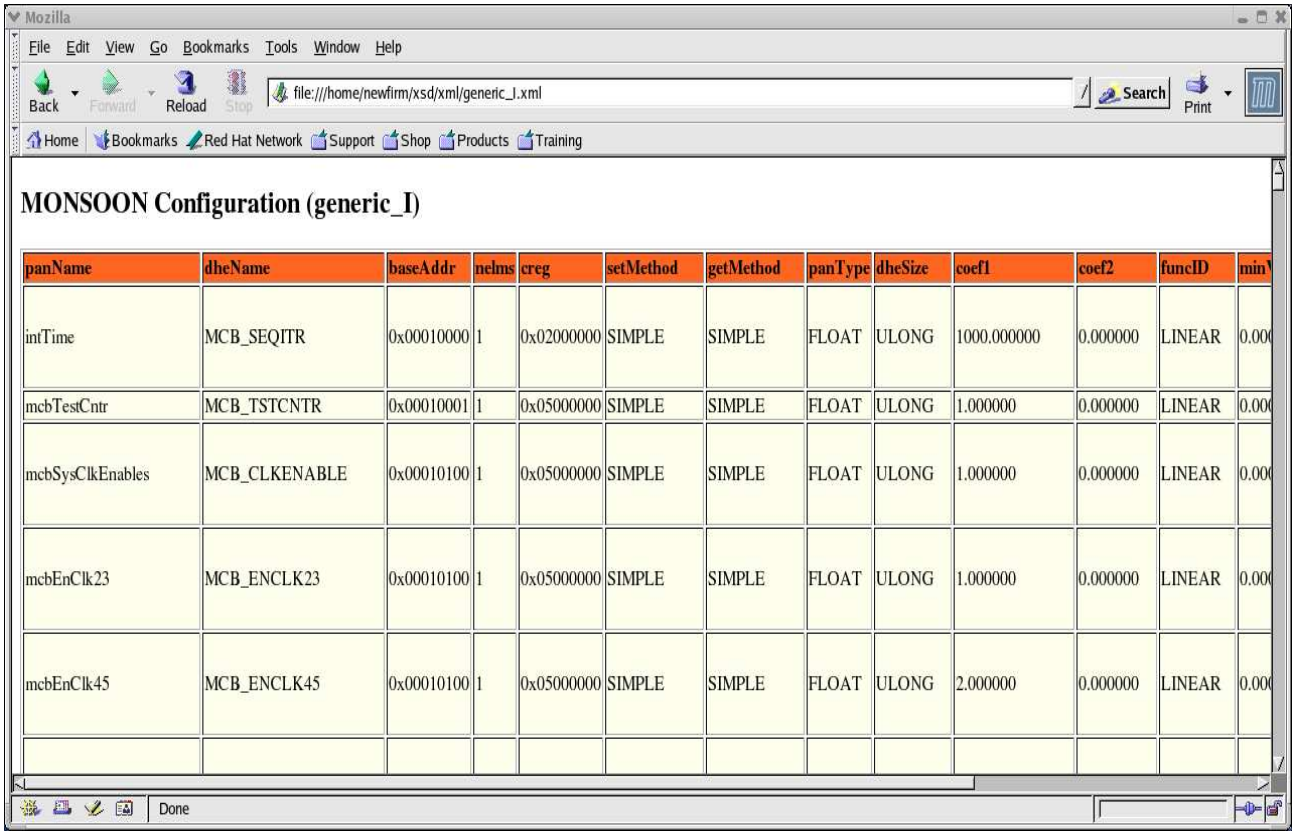
Although it is still possible to manipulate the data is a variety of ways, we have a need for simple, cross platform access to the database since hardware engineers and software engineers are known to use differing operating systems. The easiest way to handle cross-platform functionality is with Java. Thus, we are developing a Java front-end to the database that will allow suitable staff to access the records and update them as required. Although not complete, an interim, prototype interface is shown in Figure 3 on page 10.

ACKNOWLEDGMENTS

We would like to acknowledge the whole MONSOON team from NOAO-N and NOAO-S in achieving first bit, first byte and first light (for Orion, Aladdin and CCD detectors) during the period September 2003 and May 2004—and, of course, for testing the software!

REFERENCES

1. R. G. Autry, R. G. Probst, B. M. Starr, K. M. Abdel-Gawad, R. D. Blakley, P. N. Daly, R. Dominguez, E. A. Hileman, M. Liang, E. T. Pearson, R. A. Shaw and D. Tody, 2002, *NEWFIRM: the wide-field IR imager for NOAO 4-m telescopes*, Proc. SPIE Vol. 4841, Instrument Design and Performance for Optical/Infrared Ground-based Telescopes, Masanori Iye, Alan F. Moorwood, Eds. pp525-539.



panName	dheName	baseAddr	nelms	creg	setMethod	getMethod	panType	dheSize	coef1	coef2	funcID	minV
intTime	MCB_SEQITR	0x00010000	1	0x02000000	SIMPLE	SIMPLE	FLOAT	ULONG	1000.000000	0.000000	LINEAR	0.000
mcbTestCntr	MCB_TSTCNTR	0x00010001	1	0x05000000	SIMPLE	SIMPLE	FLOAT	ULONG	1.000000	0.000000	LINEAR	0.000
mcbSysClkEnables	MCB_CLKENABLE	0x00010100	1	0x05000000	SIMPLE	SIMPLE	FLOAT	ULONG	1.000000	0.000000	LINEAR	0.000
mcbEnClk23	MCB_ENCLK23	0x00010100	1	0x05000000	SIMPLE	SIMPLE	FLOAT	ULONG	1.000000	0.000000	LINEAR	0.000
mcbEnClk45	MCB_ENCLK45	0x00010100	1	0x05000000	SIMPLE	SIMPLE	FLOAT	ULONG	2.000000	0.000000	LINEAR	0.000

Figure 2. MONSOON CSV Data Displayed via XML in Web-browser.

2. R. G. Probst, N. Gaughan, G. Chisholm, P. N. Daly, E. A. Hileman, M. Hunten, M. Liang, K. M. Merrill, and J. Penegor, 2004, *Project Status of NEWFIRM: the wide-field infrared camera for NOAO 4-m telescopes*, Proc. SPIE Vol. 5492, Ground-based Instrumentation for Astronomy, Masanori Iye, Alan F. Moorwood, Eds. (in press).
3. N. C. Buchholz and P. N. Daly, 2004, *The Generic Pixel Server Dictionary*, Proc. SPIE Vol. 5496, Advanced Software, Control, and Communication Systems for Astronomy, Hilton Lewis, Gianni Raffi, Eds. (this volume).
4. N. C. Buchholz and P. N. Daly, 2004, *The MONSOON Generic Pixel Server Software Design*, Proc. SPIE Vol. 5496, Advanced Software, Control, and Communication Systems for Astronomy, Hilton Lewis, Gianni Raffi, Eds. (this volume).
5. P. N. Daly and N. C. Buchholz, 2004, *The MONSOON Implementation of the Generic Pixel Server*, Proc. SPIE Vol. 5496, Advanced Software, Control, and Communication Systems for Astronomy, Hilton Lewis, Gianni Raffi, Eds. (this volume).

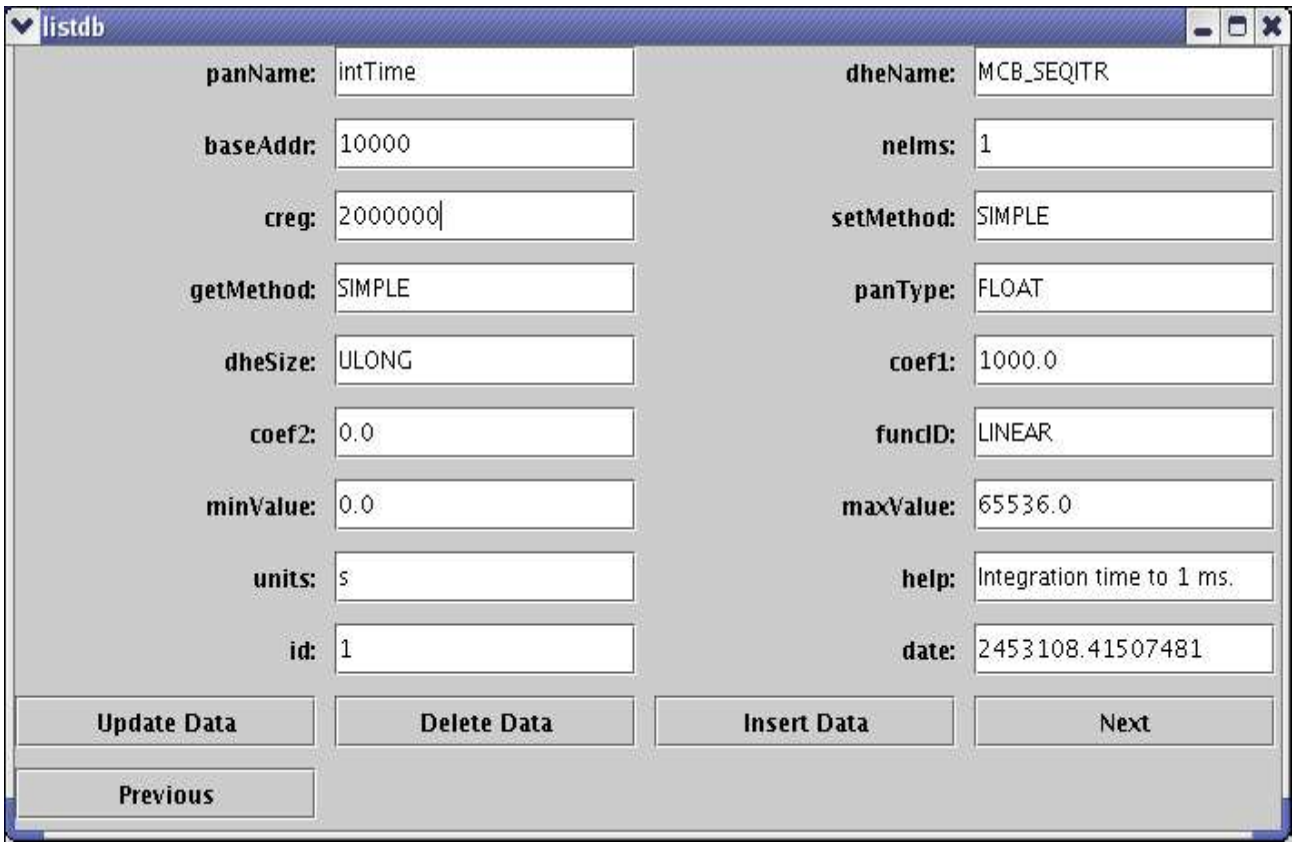


Figure 3. Interim Java Interface to the PostGreSQL Configuration Database.