

***SNS***  
***Diagnostics Controls***

**LabVIEW and C/C++  
Interface to  
DatabaseAccess DLL  
Reference**

By: Lisa Day  
day@lanl.gov

January 23, 2002

# **1 Basics of SNS BPM Control Systems**

The SNS Diagnostics Control System for the BPMs is based on PCI data acquisition and control through LabVIEW Virtual Instruments running under Windows 2000.

LabVIEW interfaces with the PCI card through calls to a Win32 acquisition DLL, which uploads data from the PCI card into LabVIEW local acquisition variables. After the data and timestamp is acquired, LabVIEW calls the DatabaseAccess DLL to update the controls database, which is shared between all connected client processes. Both of these DLLs have LabVIEW interface sub-VI's available for easily integrating the calls into main programs.

## **2 Overview of the DatabaseAccess DLL**

The DatabaseAccess DLL, da.dll, maintains a memory resident database holding control system data and accessible by client processes. The database contains control points and acquired data for the local system. The DLL controls updates and retrievals within database records providing clients with asynchronous access to the database.

This document is intended to serve as a reference for interfacing client processes, LabVIEW in particular, to the DatabaseAccess DLL. The following sections describe the requirements for the integration of the DLL with LabVIEW sub-VI's and C/C++ calls.

## **3 Defining the Database**

To minimize the operator's complexity in defining the database, an Excel spreadsheet, "daDbaseDef.xls", is formatted for simple entry of record definitions. The database is to be defined using the formatted Excel spreadsheet (Figure 1) as a template and then saved to the tab-delimited text file "daDbaseDef.txt". A copy of the text file should be placed in the c:\\daDLL directory.

	Total_recs	4			
	Total_data_pts	31902			
	EPICS_NAME	CLIENT_NAME	TYPE	NELM	DEFAULT_VAL
RECORD	Bpm1_control	control	ao	1	0
RECORD	Bpm1_acq_wid	acquisition_width	ao	1	50
RECORD	Bpm1_filtered	filtered_data	ao	15900	0
RECORD	Bpm1_raw	raw_data	ao	16000	0

**Figure 1 : Formatted Excel template file for database definition.**

NOTE: Do not modify or rearrange any cells contained in the spreadsheet except those in the rows following the key words “RECORD”. To do so could cause the DLL to fail during initialization.

DatabaseAccess parses the file based on key words. The first two rows in the spreadsheet are formatted to automatically calculate the number of records and data points contained within the database. The DLL determines the size for its memory maps based on these values. The next row contains the record fields supported. The DLL loads the records as defined by an operator by parsing the remaining rows. For each record loaded, the DLL assigns an index number (INDX) for directly accessing that record. This index can be retrieved and used to reference records and often increases performance.

For more detail about how the DLL creates and loads its memory maps, see the DatabaseAccess DLL documentation (which doesn't exist).

## **4 Exported DLL Calls**

### **4.1 General Features**

The DatabaseAccess DLL exports functions for accessing the database record fields and their values. These functions are callable by any DatabaseAccess client process.

In short, the exported functions described throughout this document support the following tasks:

- Updating and retrieving single point data and timestamps
- Updating and retrieving multipoint waveform data and timestamps
- Retrieving specific record field values
  - Client reference identification (LabVIEW variable name)
  - EPICS process variable name
  - DLL assigned index number
  - Number of data elements supported for record

For ease of integration, the DLL is able to identify records with any one of three *reference id* fields:

EPICS\_NAME: PV name to be served by Channel Access.

CLIENT\_NAME: Local LabVIEW, or other client, name.

INDX: Assigned by DLL during database load.

Routines are available to retrieve the id's given one known reference. Retrieving the INDX field during initialization and then using the INDX number during run-time is the most time efficient method for referencing any database record.

## **4.2 C++ Exported Function Declarations**

The following is a list of the exported functions from the DatabaseAccess DLL, including parameters and return values.

The six functions following provide general information about the DLL, some lists of database contents, and available commands. They are primarily used for debugging from the command prompt. Although they do not cause the DLL to modify its memory maps, I/O routines, such as printf, are called:

```
void help();
double about ();
void clientList ();
void epicsList ();
void dbList ();
void mutexList ();
```

The 'get' and 'put' functions following are exported to provide asynchronous access to the database:

```
short epicsPut (char *recName, double *newVal, double *tstamp, int nelm);
short clientPut (char *recName, double *newVal, double *tstamp, int nelm);
short dbPut (short *index, double *newVal, double *tstamp, int nelm);
short epicsGet (char *recName, double *newVal, double *tstamp, int nelm);
short clientGet (char *recName, double *newVal, double *tstamp, int nelm);
short dbGet (short *dbIndex, double *newVal, double *tstamp, int nelm);
short epicsGetNelm (char *recName, int *nelm);
short clientGetNelm (char *recName, int *nelm);
short dbGetNelm (short *index, int *nelm);
short clientGetRefs(char *clientName, char *epicsName, short *index);
short dbGetRefs (char *clientName, char *epicsName, short *index);
```

## **4.3 LabVIEW Interface VI's**

A set of sub-VI's has been configured with calls to the exported functions for ease of integration into LabVIEW programs. A LabVIEW programmer can insert these sub-VI's into his program and wire, or link, local variables as required.

As previously mentioned in 4.1, the DLL identifies records by any one of three *reference id* fields. Using the INDX field whenever possible is recommended to directly index records and reduce search time.

The following sections describe the LabVIEW sub-VI's, including the VI's icon and the wiring links required for the DLL call. Note that LabVIEW name is specified when referring to the CLIENT\_NAME field.

### **Common Attributes Shared by All sub-VI's**

All sub-VI calls maintain the same return value convention.

➤ RETURN VALUE:

- Error status – greater than zero (>0) for success

### **GET Reference Identification Fields**

Retrieve the two unknown reference id fields for the record specified.



Given the LabVIEW variable name, or other client reference, retrieve the EPICS PV name and the DLL INDX number.

➤ INPUT:

- LabVIEW variable name (client reference)

➤ OUTPUT:

- EPICS process variable name
- DLL index number



Given the DLL INDX number, retrieve the EPICS PV name and LabVIEW name (or client reference).

➤ INPUT:

- DLL index number

➤ OUTPUT:

- EPICS process variable name
- LabVIEW variable name

### **GET Single Point VALues**

Retrieve the record's single point data value and associated time stamp.

#### **Common Output**

➤ OUTPUT:

- Data VAL
- Time stamp

 **DaGetValByName**

- INPUT:
- LabVIEW variable name

 **DaGetValByIndx**

- INPUT:
- DLL index number

**PUT Single Point VALues**

 **DaPutValByName**

- INPUT:
- LabVIEW variable name
  - Data VAL
  - Time stamp

 **DaPutValByIndx**

- INPUT:
- DLL index number
  - Data VAL
  - Time stamp

**GET Multi-point Data Array (or Waveform)**

Retrieve an  $N$  element data array and its associated time stamp. Client processes can request any number of data points up to the default NELM specified in the database definition file. The DLL will retrieve the requested number of elements unless that number:

- ➔ Equals zero ( $=0$ )
- ➔ Less than zero ( $<0$ )
- ➔ Greater than NELM ( $>NELM$ )

Under these conditions, the DLL will retrieve its default NELM for that record.

**Common Attributes**

- OUTPUT:

- Data array
- Time stamp

NOTE: Memory access errors may occur if initialized arrays passed in are not large enough to contain all of the retrieved data.

**Get Array** **DaGetArrayByName**

➤ INPUT:

- LabVIEW variable name
- Number of elements to retrieve ( $N$ )
- Initialized array of size  $N$

**Get Array** **DaGetArrayByIndx**

➤ INPUT:

- DLL index number
- Number of elements to retrieve ( $N$ )
- Initialized array of size  $N$

**PUT Multi-point Data Array**

Retrieve an  $N$  element data array and its associated time stamp. Client processes can request any number of data points up to the default NELM specified in the database definition file. The DLL will retrieve the requested number of elements unless that number:

- ➔ Equals zero ( $=0$ )
- ➔ Less than zero ( $<0$ )
- ➔ Greater than NELM ( $>NELM$ )

Under these conditions, the DLL will retrieve its default NELM for that record.

**PUT Array** **daPutArrayByName**

➤ INPUT:

- LabVIEW variable name
- Array containing acquired data
- Time stamp

**PUT Array** **daPutArrayByIndx**

➤ INPUT:

- DLL index number

- Array containing acquired data
- Time stamp

### **GET Default Number of Elements (NELM)**

The NELM field for a record is most commonly needed for a parameter when retrieving and updating data arrays.

#### **Common Attributes**

➤ OUTPUT:

- Record's default NELM

#### **DaGetNelmByName**

➤ INPUT:

- LabVIEW variable name

#### **DaGetNelmByIndx**

➤ INPUT:

- DLL index number