IPm Library Function Calls
These library functions are used within applications to exchange I/O with an IPm station's I/O registers. Upcoming is a list of the function prototypes. Each function topic includes a C example.

Refer to "Using the IPm library function calls in your program" (see pg 24) for more information on using these library calls.

Refer to the supplied sample program, oem_iodb.c, which uses these library function calls.

IPm library function descriptions

I/O Updating Functions

IODBSetTag (See page 2)	set a tag to correspond to a register or block of registers
IODBRead (See page 3)	read one or more consecutive I/O registers
IODBReadTag (See page 4)	read one register referenced by I/O tag name
IODBWrite (See page 5)	write values to consecutive I/O registers
IODBWriteMask (See page 6)	write to select registers
IODBWriteTag (See page 7)	write to one register referenced by I/O tag name

Advanced Driver Functions

IODBGetNextTag (See page 8)	used to build a list of valid I/O tag names (Sixtags)
IODBGetNextType (See page 9)	used to build a list of valid I/O types
IODBGetTag (See page 10)	returns facts about one I/O tag name (Sixtag)
IODBGetType (See page 12)	returns facts about an I/O type
IODBGetTypeRange (See page 14)	tells the range of available I/O registers
IODBVersion (See page 15)	returns the version number of the library that is currently running
IODBGetFile (See page 16)	returns information about the .6pj file

I/O Status Functions

IODBGetDescription (See page 17)	returns the description for an I/O point as defined in the I/O Tool Kit.
IODBGetFormat (See page 18)	returns the format of an analog input, as defined in the project file.
IODBMinMax (See page 19)	returns the engineering units and the minimum and maximum for both the raw reading and scaled value for an analog I/O register.
IODBMinMaxTag (See page 21)	returns the engineering units and the minimum and maximum for both the raw reading and scaled value for an analog I/O tag name.
IODBScale (See page 22)	returns the engineering units and conversion factors for an analog I/O register.
IODBScaleTag (See page 23)	returns the engineering units and conversion factors for an analog I/O tag name.

Note: The IODBSetTag, IODBReadTag, IODBWriteTag, IODBGetNextTag, IODBGetTag, IODBMinMaxTag and IODBScaleTag functions require that the "project tag list" be loaded into the IPm station. Make sure the "Load project tag list" checkbox in the "Files to load" window in the I/O Tool Kit is selected, then load the configuration into the IPm station.

IODBSetTag

Use this function...

to set a tag to correspond to a register or block of registers.

TagName specifies the Sixtag name (prefix and tag name), **TypeNum** specifies a valid type number (0-126), **Addr** specifies the starting I/O database address of the registers directly, and **NumRegs** specifies the number of registers to write.

Return Values:

ENOERROR	- success
EOUTOFRANGE	- Addr + NumRegs exceeded I/O type allocation
EINVALIDTYPE	- invalid type number

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBSetTag(
  const char     *TagName,  /* Sixtag (prefix and tag name) */
  USHORT         TypeNum,   /* I/O type number */
  USHORT         Addr,      /* starting address */
  USHORT         NumRegs    /* number of I/O points */
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;

...

/*
 * Assign the tagname, "Station1.Tag1", to 10 registers
 * of type 0 (Analog In), starting at address 0.
 */
eCode = IODBSetTag("Station1.Tag1", 0, 0, 10);
```

IODBRead

Use this function...

to read one or more consecutive I/O values from the I/O database.

TypeNum specifies a valid type number (0-126), **NumRegs** specifies the number of registers to read, and **pbuff** is a pointer to a buffer where the I/O is returned. **Addr** specifies the starting I/O database address of the registers directly. **StaName** pointer is not used, pass NULL.

Return Values:

ENOERROR	- success
EOUTOFRANGE	- Addr + NumRegs exceeded I/O type allocation
ENOSTATION	- invalid Addr or TypeNum for station
EINVALIDTYPE	- invalid type number

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBRead(
  USHORT         TypeNum,    /* I/O type number */
  USHORT         Addr,       /* starting address */
  USHORT         NumRegs,    /* number of I/O points */
  void           *pbuff,     /* pointer to return buffer for I/O */
  const char     *StaName    /* pointer is not used, pass NULL*/
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;
USHORT   Analogs[10];
BYTE     Discretes[2];

...

/*
 *  Reads 10 integers of type 0 (Analog In) directly from
 *  the I/O database starting at address 3.
 */
eCode = IODBRead(0, 3, 10, Analogs, NULL);

/*
 *  Reads 10 discrete registers of type 10 (Discrete In)
 *  directly from the I/O database starting at address 3.
 *  Note: Addr does not need to be on a byte boundary, nor does
 *  NumRegs need to be a multiple of 8.
 */
     eCode = IODBRead(10, 3, 10, Discretes, NULL);
```

IODBReadTag

Use this function...

to read a value from one I/O register in the I/O database at the address associated with the specified I/O Tag Name (Sixtag).

TagName specifies the Sixtag name (prefix and tag name), **pBuf** is a pointer to a buffer where the I/O is returned, **BufSize** is the size of the return buffer in bytes. The Sixtag data type is returned in **DataType**, which can be set to NULL if no return information is desired. The number of registers is returned in **NumRegs**, which can be set to NULL if no return information is desired. Tag names may be delineated by a '.' or a '_'. For example, both "Station1.Tag1" and "Station1_Tag1" are valid Sixtag designations.

Return Values:

ENOERROR	- success
ETAGNOTFOUND	- Sixtag not found
EDUPLICATETAG	- Sixtag has multiple definitions
ENOSTATION	- Station address of Sixtag I/O is invalid

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBReadTag(
  const char     *TagName,  /* Sixtag (prefix and tag name) */
  void           *pBuf,     /* pointer to return buffer for I/O */
  USHORT         BufSize,   /* size of return buffer (in bytes) */
  USHORT         *DataType, /* data type */
  USHORT         *NumRegs   /* number of I/O points */
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;
USHORT   DataType;
USHORT   Analog;
USHORT   NumRegs;

...

eCode = IODBReadTag("Station1.Tag1",&Analog,sizeof(Analog),
                         &DataType, &NumRegs);
```

IODBWrite

Use this function...

to write one or more consecutive I/O values into the I/O database.

TypeNum specifies a valid type number (0-126), **NumRegs** specifies the number of registers to write, and **pbuff** is a pointer to a buffer of I/O to write. **Addr** specifies the starting I/O address of the registers directly. **StaName** pointer is not used, pass NULL.

Return Values:

ENOERROR	- success
EOUTOFRANGE	- Addr + NumRegs exceeded I/O type allocation
ENOSTATION	- invalid Addr or TypeNum for station
EINVALIDTYPE	- invalid type number

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBWrite(
  USHORT        TypeNum,   /* I/O type number */
  USHORT        Addr,      /* starting address */
  USHORT        NumRegs,   /* number of I/O points */
  void          *pbuff,    /* pointer to buffer to write */
  const char    *StaName   /* Pointer not used, pass NULL
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;
USHORT   Analogs[] = { 0,100,200,300,400,500,600,700,800,900 };
BYTE   Discretes[] = { 0x31, 0x03 };

...

/*
 *  Writes 10 integers of type 1 (Analog Out) directly into
 *  the I/O database starting at address 3.
 */
eCode = IODBWrite(1, 3, 10, Analogs, NULL);

/*
 *  Writes 10 discrete points of type 11 (Discrete Out) directly into
 *  the I/O database starting at address 3. Sets I/O
 *  addresses 3, 7, 8, 11, and 12 ON, and sets I/O addresses 4, 5,
 *  6, 9, and 10 OFF. Note: Addr does not need to be on
 *  a byte boundary, nor does NumRegs need to be a
 *  multiple of 8.
 */
eCode = IODBWrite(11, 3, 10, Discretes, NULL);
```

IODBWriteMask

Use this function...

to write selected I/O values into the I/O database.

TypeNum specifies a valid type number (0-126), **NumRegs** specifies the number of registers to write, and **pbuff** is a pointer to a buffer of I/O to write. **pMask** specifies the write mask (a 1 indicates the point will be updated, and a 0 indicates the point will remain unchanged). The bit order of the mask is as follows: the least-significant bit of the first byte of **pMask** corresponds to the register at address **Addr**. The bit positions continue in ascending order to the most-significant bit and then continues on to the least-significant-bit of the next byte. StaName pointer is not used, pass NULL.

Return Values:

ENOERROR	- success
EOUTOFRANGE	- Addr + NumRegs exceeded I/O type allocation
ENOSTATION	- invalid Addr or TypeNum for station
EINVALIDTYPE	- invalid type number

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBWriteMask(
  USHORT         TypeNum,    /* I/O type number */
  USHORT         Addr,       /* starting address */
  USHORT         NumRegs,    /* number of I/O points */
  void           *pbuff,     /* pointer to buffer to write */
  BYTE           *pMask,     /* mask of points to write */
  const char     *StaName    /* pointer not used, pass NULL */
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;
BYTE Discretes[] = { 0xff, 0xff };
BYTE mask[]      = { 0x0f, 0x00 };

...

/*
 *  Writes 10 discrete registers of type 11 (Discrete Out) directly
 *  into the I/O database starting at address 3. Only
 *  addresses 3, 4, 5, 6, will actually updated (as specified by the
 *  mask) the remaining points will not be changed. Note: Addr does
 *  not need to be on a byte boundary, nor does NumRegs
 *  need to be a multiple of 8.
 */
eCode = IODBWriteMask(11, 3, 10, Discretes, mask, NULL);
```

IODBWriteTag

Use this function...

to write a value to one I/O register in the I/O database at the address associated with the specified I/O Tag Name (Sixtag).

TagName specifies the Sixtag name (prefix and tag Name) and **pBuf** is a is a pointer to a buffer of I/O to write. Tag names may be delineated by a '.' or a '_'. For example, both "Station1.Tag1" and "Station1_Tag1" are valid Sixtag designations.

Return Values:

ENOERROR	- success
ETAGNOTFOUND	- Sixtag not found
EDUPLICATETAG	- Sixtag has multiple definitions
ENOSTATION	- Station address of Sixtag I/O is invalid

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBWriteTag(
  const char     *TagName,  /* Sixtag name (prefix and tag name) */
  void           *pBuf      /* pointer to buffer to write */
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;
USHORT   Analog = 900;

...
/* Write Value 900 to tag "Station1.Tag1" */
eCode = IODBWriteTag("Station1.Tag1", &Analog);
```

IODBGetNextTag

Use this function...

to query the I/O database for information about the next valid I/O Tag Name (Sixtag) in the database. Successive calls to this function will return a list of all valid Sixtags.

Gets Sixtag information identified by the next valid Sixtag following ListPos, then sets ListPos to the position of the next entry. Use IODBGetNextTag in a forward iteration loop by establishing the initial position with a ListPos of -1. When no valid Sixtags remain in the list, IODBGetNextTag will return ETAGNOTFOUND. Any of the parameters used to return query information, **TagName**, **StaName**, **TypeNum**, **DataType**, **TypeSize**, and **RTUAddr** can be set to NULL if no return information in desired.

If **match** is NULL all valid Sixtags will be returned. If **match** is a string ending with '.' or '_', only valid Sixtags with the specified prefix will be returned. If **match** is a string that does not end with '.' or '_', it is assumed **match** specifies the actual station name, and only valid Sixtags with the specified station name will be returned.

Return Values:

ENOERROR	- success
ETAGNOTFOUND	- no valid Sixtags remain in list

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetNextTag(
  short    *ListPos,  /* list iteration variable */
  char     *match,    /* tag search variable */
  char     *TagName,  /* Sixtag (prefix and tag name) */
  char     *StaName,  /* pointer is not used, pass NULL */
  USHORT   *TypeNum,  /* I/O type number */
  USHORT   *DataType, /* data type */
  USHORT   *TypeSize, /* type size (bytes) */
  USHORT   *RTUAddr   /* I/O station address */
);
```

C Example:

```
#include <iodb.h>

USHORT   TypeNum;
USHORT   DataType;
USHORT   TypeSize;
USHORT   RTUAddr;
char     TagName[30];
short    ListPos = -1;
IODBerr eCode = ENOERROR;

while (eCode == ENOERROR) {
  eCode = IODBGetNextTag(&ListPos, NULL, TagName, NULL, &TypeNum,
          &DataType, &TypeSize, &RTUAddr);

  if (eCode == ENOERROR)
    fprintf(stdout, "Sixtag %s Information: %u, %u, %u, %u",
            TagName, TypeNum, DataType, TypeSize, RTUAddr);
}
```

IODBGetNextType

Use this function...

to query the I/O database for information about the next valid I/O type in the configuration. Successive calls to this function will return a list of all valid I/O types.

Gets I/O type information identified by the next valid type following **ListPos**, then sets **ListPos** to the position of that entry. Use **IODBGetNextType** in a forward iteration loop by establishing the initial position with a **ListPos** of -1. When no valid I/O types remain in the list, **IODBGetNextType** will return EINVALIDTYPE. Any of the parameters used to return query information, **TypeNum**, **DataType**, **TypeSize**, **NumAlloc**, and **TypeName**, can be set to NULL if no return information is desired. See **IODBGetType** for return parameter information.

Pass NULL as **StaName** as the pointer is not used.

Return Values:

ENOERROR	- success
EINVALIDTYPE	- no valid I/O types remain in list

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetNextType(
  short          *ListPos,  /* list iteration variable */
  USHORT         *TypeNum,  /* type number */
  USHORT         *DataType, /* data type */
  USHORT         *TypeSize, /* type size (bytes) */
  USHORT         *NumAlloc, /* number of registers allocated */
  char           *TypeName, /* type name */
  const char     *StaName   /* pointer not used, pass NULL */
);
```

C Example:

```
#include <iodb.h>

USHORT   DataType;
USHORT   TypeSize;
USHORT   NumAlloc;
USHORT   TypeNum;
char     TypeName[21];
short    ListPos = -1;
IODBerr eCode = ENOERROR;

...

while (eCode == ENOERROR) {
  eCode = IODBGetNextType(&ListPos, &TypeNum, &DataType,
                          &TypeSize, &NumAlloc, TypeName, NULL);

  if (eCode == ENOERROR) {
    fprintf(stdout,"Type %u Information: (%s), %u, %u, %u",
              TypeNum, TypeName, DataType, TypeSize, NumAlloc);
  }
}
```

IODBGetTag

Use this function...

to query the I/O database for information about a particular I/O register associated with the specified I/O Tag Name (Sixtag).

Any of the parameters used to return query information, **TypeNum**, **DataType**, **TypeSize**, **NumRegs**, and **RTUAddr** can be set to NULL if no return information is desired. StaName is not used and should be passed NULL. Tag names may be delineated by a '.' or a '_'. For example, both "Station1.Tag1" and "Station1_Tag1" are valid Sixtag designations.

Return Values:

ENOERROR	- success
ETAGNOTFOUND	- Sixtag not found
EDUPLICATETAG	- Sixtag has multiple definitions

Values returned in **DataType** (defined in iodb.h):

ANALOGtype
DISCRETEtype
BYTEtype
LONGtype
FLOATtype
DOUBLEtype
USERtype

Values returned in **TypeSize**:

2 (Analog)
0 (Discrete)
1 (Byte)
4 (Long)
4 (Float)
8 (Double)
1-220 (User-Type)

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetTag(
  const char     *TagName,  /* Sixtag (prefix and tag name) */
  char           *StaName,  /* pointer not used, pass NULL */
  USHORT         *TypeNum,  /* I/O type number */
  USHORT         *DataType, /* data type */
  USHORT         *TypeSize, /* I/O type size (bytes) */
  USHORT         *Numregs,  /* Number of I/O points */
  USHORT         *RTUAddr   /* RTU address of register */
);
```

C Example:

```
#include <iodb.h>

USHORT   TypeNum;
USHORT   DataType;
USHORT   TypeSize;
USHORT   NumRegs;
USHORT   RTUAddr;
IODBerr eCode;

...

eCode = IODBGetTag("Station1.Tag1", NULL, &TypeNum, &DataType,
        &TypeSize, &NumRegs, &RTUAddr);

if (eCode == ENOERROR) {
  fprintf(stdout, "Station1.Tag1 Information: %u, %u, %u, %u, %u",
          TypeNum, DataType, TypeSize, NumRegs, RTUAddr);
}
```

IODBGetType

Use this function...

to query the I/O database for information about a particular I/O type.

Any of the parameters used to return query information, **DataType**, **TypeSize**, **NumAlloc**, and **TypeName**, can be set to NULL if no return information is desired.

Return Values:

ENOERROR	- success
EINVALIDTYPE	- Invalid type number

Values returned in **DataType** (defined in iodb.h):

ANALOGtype
DISCRETEtype
BYTEtype
LONGtype
FLOATtype
DOUBLEtype
USERtype

Values returned in **TypeSize**:

2 (Analog)
0 (Discrete)
1 (Byte)
4 (Long)
4 (Float)
8 (Double)
1-220 (User-Type)

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetType(
  USHORT   TypeNum,   /* I/O type number */
  USHORT   *DataType, /* data type */
  USHORT   *TypeSize, /* I/O type size (bytes) */
  USHORT   *NumAlloc, /* number of registers allocated */
  char     *TypeName  /* I/O type name */
);
```

C Example:

```
#include <iodb.h>

USHORT   DataType;
USHORT   TypeSize;
USHORT   NumAlloc;
char     TypeName[21];
IODBerr eCode;

...

eCode = IODBGetType(1,&DataType,&TypeSize,&NumAlloc,TypeName);

if (eCode == ENOERROR) {
  fprintf(stdout, "Type 1 Information: (%s), %u, %u, %u", TypeName,
          DataType, TypeSize, NumAlloc);
}
/*
 *  For example, this would display (using defaults):
 *  "Type 1 Information: Analog Out,1,2,1024"
 *
 */
```

IODBGetTypeRange

Use this function...

to query the I/O database for information about the configuration range (first and last available registers) of an I/O type.

StaName is not used and should be passed NULL. Undefined station addresses between the first and last address (non-contiguous) will be ignored.

Return Values:

ENOERROR - success
ENOSTATION - invalid TypeNum for station
EINVALIDTYPE - invalid type number

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetTypeRange(
  USHORT         TypeNum,   /* I/O type number */
  const char     *StaName,  /* pointer not used, pass NULL */
  USHORT         *MinAddr,  /* first address */
  USHORT         *MaxAddr   /* last address  */
);
```

C Example:

```
#include <iodb.h>

USHORT   MinAddr;
USHORT   MaxAddr;
IODBerr eCode;

...

eCode = IODBGetTypeRange(0, NULL, &MinAddr, &MaxAddr);

if (eCode == NOERROR)
  fprintf(stdout, "Address Range: %u - %u", MinAddr, MaxAddr);
```

IODBVersion

Use this function...

to get the version number of the DLL that is currently running.

This function returns an unsigned integer representing the major and minor version information of the DLL that is currently executing.

C Prototype and Example

C Prototype:

```
#include <iodb.h>
USHORT IODBVersion();
```

C Example:

```
#include <iodb.h>

USHORT ver;

ver = IODBVersion();
fprintf(stdout, "Version = %u.%u",((ver >> 8) & 0xff), (ver & 0x00ff));
```

IODBGetFile

Use this function...

to get information about the disk file (project configuration) associated with the IODB that is currently running.

Any of the parameters used to return query information, **PathName**, **ConfigName**, and **ConfigDate** can be set to NULL if no return information is desired. If no IODB configuration is currently running, **PathName** and **ConfigName** return an empty string and **ConfigDate** returns 0.

C Prototype and Example

C Prototype:

```
#include <iodb.h>
void IODBGetFile(
  char     *PathName,       /* full path name of project file */
  char     *ConfigName,     /* IODB configuration name */
  time_t   *ConfigDate           /* time_t of last modification */
);
```

C Example:

```
#include <iodb.h>
#include <string.h>
#include <time.h>

char   PathName[256];
char   ConfigName[21];
time_t ConfigDate;

...

IODBGetFile(PathName, ConfigName, &ConfigDate);

if (strlen(PathName) == 0)
  fprintf(stderr,"No configuration currently loaded");
else {
  fprintf(stdout, "File Info: %s (%s) %s",
          PathName, ConfigName, ctime(&ConfigDate));
}
/*
 *  For example, the above might display:
 *  File Info: /etc/stacfg/project.6pj (project) Thu Feb 07 12:00:00 2003
 */
```

IODBGetDescription

Use this function...

to get the user-defined description (comment text) of an I/O register in an IPm station. The description is defined using the I/O Tool Kit and is stored in the project file associated with the station.

Gets the description of the particular I/O register identified by the station name, I/O type, and register address. Descriptions are limited to 40 characters, plus a null character.

Return Values:

ENOERROR	- success
EINVALIDTYPE	- type specified is not analog or discrete I/O
EREGISTERNOTFOUND	- address is out of range for station

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetDescription(
  const char     *StaName,        /* pointer not used, pass NULL */
  USHORT         TypeNum,         /* type of register */
  USHORT         Addr,            /* address of register */
  Char           *Description     /* returned description */
);
```

C Example:

```
#include <iodb.h>

IODBerr eCode;
USHORT   TypeNum;
USHORT   Addr;
char     Description[40];


TypeNum = 10;
Addr = 0;

eCode = IODBGetDescription(NULL, TypeNum, Addr, &Description);

if (eCode)
  fprintf(stdout, "Error = %u\n", eCode);
else
  fprintf(stdout, "Description = %s\n", Description);
```

IODBGetFormat

Use this function...

to get the format for an analog input register in an IPm station as defined by the I/O Tool Kit software and stored in the project file associated with the station.

Gets the integer format of the particular analog input register identified by the station name, I/O type (always 0), and register address. The format of the analog input registers are loaded from the project file upon initialization of the I/O database. Station Name is not used, set it to NULL.

Return Values:

ENOERROR	- success
ENOTANALOGTYPE	- type specified is not analog input
EOUTOFRANGE	- address is out of range for entire I/O database

Values returned in **Format** (defined in iodb.h):

NOFORMAT	- no format found
SFORMAT	- signed integer
UFORMAT	- unsigned integer
LFORMAT	- signed long

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBGetFormat(
  const char     *StaName,  /* pointer not used, pass NULL */
  USHORT         TypeNum,   /* type of register */
  USHORT         Addr,      /* address of register */
  BYTE           *Format    /* returned format */
);
```

C Example:

```
#include <iodb.h>

IODBerr  eCode;
USHORT   TypeNum = 0;
USHORT   Addr;
BYTE     Format;

Addr = 0;
eCode = IODBGetFormat(NULL, TypeNum, Addr, &Format);

if (eCode)
  fprintf(stdout, "Error = %u\n", eCode);
else
  fprintf(stdout, "Format = %d\n", Format);
```

IODBMinMax

Use this function...

to get the engineering units, and both the raw and scaled minimum and maximum for an analog register in an IPm station as defined by the I/O Tool Kit software and stored in the project file associated with the station.

This function gets the engineering units and the raw and scaled minimum and maximum of an analog I/O register identified by the station name, I/O type, and register address. This function looks into the I/O database for the name of the project file associated with the station. It then returns the raw and scaled minimum and maximum, as well as the engineering units character string associated with the register.

Any of the parameters used to return query information, **RawMin**, **RawMax**, **ScaleMin**, **ScaleMax**, and **Units** can be set to NULL if no return information is desired. If more than one station definition exists in the project, it will return information from the first one found.

Return Values:

ENOERROR	- success.
ENOTANALOGTYPE	- register is not an analog type.
EREGISTERNOTFOUND	- register not found.
EFILENOTFOUND	- project file not found.
EFILEACCESS	- error accessing project file.
ENOSIXNETFILE	- file does not support SIXNET configuration.

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBMinMax(
  const char     *StaName,  /* pointer not used, pass NULL */
  USHORT         TypeNum,   /* I/O type number */
  USHORT         RegAddr,   /* register address */
  long           *RawMin,   /* Minimum for raw readings */
  long           *RawMax,   /* Maximum for raw readings */
  double         *ScaleMin, /* Minimum for scaled values */
  double         *ScaleMax, /* Maximum for scaled values */
  char           *Units     /* units string */
);
```

C Example:

```
#include <iodb.h>

long      RawMin, RawMax;
double    ScaleMin, ScaleMax;
char      Units[25];
IODBerr   eCode;
USHORT    TypeNum;
USHORT    Addr;



TypeNum = 0; /* Data Type must be a 0 or 1 */
Addr = 1;

eCode = IODBMinMax(NULL, TypeNum, Addr, &RawMin, &RawMax,
        &ScaleMin, &ScaleMax, Units);

if (eCode)
  fprintf(stdout, "Error: %u\n", eCode);
else {
  fprintf(stdout, "Raw: %ld to %ld\nScaled: %lf to %lf\nUnits: %s\n",
          RawMin, RawMax, ScaleMin, ScaleMax, Units);
}
```

IODBMinMaxTag

Use this function...

to get the engineering units and both the raw and scaled minimum and maximum for an analog I/O tag in an IPm station as defined by the I/O Tool Kit software and stored in the project file associated with the station.

This function gets the engineering units and the raw and scaled minimum and maximum for an analog I/O tag identified by the its Sixtag name. This function looks into the I/O database for the name of the project file associated with the station. It then returns the raw and scaled minimum and maximum, as well as the engineering units character string associated with the register.

Any of the parameters used to return query information, **RawMin**, **RawMax**, **ScaleMin**, **ScaleMax**, and **Units** can be set to NULL if no return information is desired. If more than one station definition exists in the project, it will return information from the first one found.

Return Values:

ENOERROR	- success.
ENOTANALOGTYPE	- register is not an analog type.
EDUPLICATETAG	- Sixtag has multiple definitions.
ETAGNOTFOUND	- Sixtag not found.
EFILENOTFOUND	- project file not found.
EFILEACCESS	- error accessing project file.
ENOSIXNETFILE	- file does not support SIXNET configuration.

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBMinMaxTag(
  const char     *TagName,  /* tag name */
  long           *RawMin,   /* Minimum for raw readings */
  long           *RawMax,   /* Maximum for raw readings */
  double         *ScaleMin, /* Minimum for scaled values */
  double         *ScaleMax, /* Maximum for scaled values */
  char           *Units     /* units string */
);
```

C Example:

```
#include <iodb.h>

long      RawMin, RawMax;
double    ScaleMin, ScaleMax;
char      Units[25];
IODBerr  eCode;
char      TagName[32];

strcpy(TagName, "Demo.Current Temperature");

eCode = IODBMinMaxTag(TagName, &RawMin, &RawMax, &ScaleMin, &ScaleMax,
Units);

if (eCode)
  fprintf(stdout, "Error: %u\n", eCode);
else
  fprintf(stdout, "Raw: %ld to %ld\nScaled: %lf to %lf\nUnits: %s\n",
          RawMin, RawMax, ScaleMin, ScaleMax, Units);
```

IODBScale

Use this function...

to get the engineering units and conversion factors for an analog register in an IPm station as defined by the I/O Tool Kit software and stored in the project file associated with the station.

Gets the engineering units and conversion factors of an analog I/O register identified by the station name, I/O type, and register address. This function looks into the I/O database for the name of the project file associated with the station. It then returns the span and offset correction factors, and the engineering units character string associated with the register.

Any of the parameters used to return query information, **Span**, **Offset**, and **Units** can be set to NULL if no return information is desired. If more than one station definition exists in the project, it will return information from the first one found.

Return Values:

ENOERROR	- success.
ENOTANALOGTYPE	- register is not an analog type.
EREGISTERNOTFOUND	- register not found.
EFILENOTFOUND	- project file not found.
EFILEACCESS	- error accessing project file.
ENOSIXNETFILE	- file does not support SIXNET configuration.

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBScale(
  const char  *StaName,  /* pointer not used, pass NULL */
  USHORT      TypeNum,   /* I/O type number */
  USHORT      RegAddr,   /* register address */
  double      *Span      /* span correction factor */
  double      *Offset,   /* offset correction factor */
  char        *Units     /* units string */
);
```

C Example:

```
#include <iodb.h>

double    span, offset;
char      Units[25];
IODBerr   eCode;
USHORT    TypeNum;
USHORT    Addr;

TypeNum = 0; /* Data Type must be a 0 or 1 */
Addr = 1;

eCode = IODBScale(NULL, TypeNum, Addr, &span, &offset, Units);

if (eCode)
  fprintf(stdout, "Error: %u\n", eCode);
else
  fprintf(stdout, "Span: %lf Offset: %lf Units: %s\n", span, offset, Units);
```

IODBScaleTag

Use this function...

to get the engineering units and conversion factors for an analog I/O tag in an IPm station as defined by the I/O Tool Kit software and stored in the project file associated with the station.

Gets the engineering units and conversion factors of an analog I/O tag identified by the its Sixtag name. This function looks into the I/O database for the name of the project file associated with the station. It then returns the span and offset correction factors, and the engineering units character string associated with the register.

Any of the parameters used to return query information, **Span**, **Offset**, and **Units** can be set to NULL if no return information is desired. If more than one station definition exists in the project, it will return information from the first one found.

Return Values:

ENOERROR	- success.
ENOTANALOGTYPE	- register is not an analog type.
EDUPLICATETAG	- Sixtag has multiple definitions.
ETAGNOTFOUND	- Sixtag not found.
EFILENOTFOUND	- project file not found.
EFILEACCESS	- error accessing project file.
ENOSIXNETFILE	- file does not support SIXNET configuration.

C Prototype and Example

C Prototype:

```
#include <iodb.h>
IODBerr IODBScaleTag(
  const char     *TagName,  /* tag name */
  double         *Span,     /* span correction factor */
  double         *Offset,   /* offset correction factor */
  char           *Units     /* units string */
);
```

C Example:

```
#include <iodb.h>

double   span;
double   offset;
char     Units[25];
IODBerr eCode;
char     TagName[32];

strcpy(TagName, "demo.Current Temperature");

eCode = IODBScaleTag(TagName, &span, &offset, Units);

if (eCode)
  fprintf(stdout, "Error: %u\n", eCode);
else {
  fprintf(stdout, "Span: %lf Offset: %lf Units: %s\n", span, offset, Units);
}
```

Using the IPm Library function calls in your program

The IPm library function calls in this document make an IPm station's I/O registers readily available to C applications. These functions represent the highest abstraction level the IPm station provides to its calling clients (applications). For an application to use these library calls, IODB.H must be included in every module referencing an IPm library function.