Aspen InfoPlus.21™

Database API Manual

Version: V7.1
January 2009

Aspen Technology, Inc.
200 Wheeler Road
Burlington, MA 01803-5501
USA
Phone: +(781) 221-6400
Toll Free: (1) (888) 996-7100
http://www.aspentech.com

Contents

1 Introduction

This introductory chapter provides:

- a brief description of the information in this manual
- a description of the overall organization of this manual
- a list of related documentation
- customer support information

A routine library and header file for the C language is included in the Aspen InfoPlus.21 distribution. Any language can be used to call the supplied access routines in the library if the language allows programs to pass arguments to the routines in the same manner as the C language. Users who program in other languages will also have to have header files corresponding to the C include files distributed with Aspen InfoPlus.21. See the appropriate compiler documentation for argument passing methods.

This manual divides the Aspen InfoPlus.21 library routines into three major categories:

- access routines
- utility routines
- remote functions

Access routines access the Aspen InfoPlus.21 database and perform the following functions:

- read database fields
- write database fields
- create records
- increase the database size

Note: Access routines are discussed in the "Access Routines" chapter.

Utility routines do not access the Aspen InfoPlus.21 database. For example, a routine that compares two character strings is a utility routine, since the comparison is achieved without accessing the Aspen InfoPlus.21 database.

Note: Utility routines are discussed in the "Utility Subroutines" chapter.

Remote functions allow user-written programs to access remote Aspen InfoPlus.21 systems. These functions allow the program to:

- create records
- delete records usable
- make records unusable read any field
- write to any changeable field in a remote Aspen InfoPlus.21 database

Note: Remote access functions are discussed in the "Remote Access Functions" chapter.

The Manual

Organization

This manual contains the following:

Chapter 1 – *Introduction* – provides a brief overview of the manual and a list of related documentation.

Chapter 2 – *Installation and Setup* – provides instructions on installing and linking to the API.

Chapter 3 – *External Tasks* – discusses programs that process activated records.

Chapter 4 – *Access Routines* – provides a summary of subroutine functionality and discusses routines that give programs access to the Aspen InfoPlus.21 database.

Chapter 5 – *Utility Subroutines* – provides a summary of subroutine functionality and a more detailed description of subroutines that do not access the Aspen InfoPlus.21 database but are useful in developing Aspen InfoPlus.21 applications.

Chapter 6 – *Remote Access Functions* – provides a brief description of routines that allow a user-written program to access a remote Aspen InfoPlus.21 database.

Chapter 7 – *FORTRAN Access to the InfoPlus.21 API* – provides a brief description accessing the API from the FORTRAN programming language.

Related Documentation

In addition to this manual, a number of other manuals are provided to help users learn and use Aspen InfoPlus.21. The documentation set consists of the following manuals and help files:

Manuals/Help Files

Aspen InfoPlus.21 Installation Manual

Aspen InfoPlus.21 Database API Manual

Aspen InfoPlus.21 Database User's Manual

Aspen InfoPlus.21 Database Developer's Manual

Aspen InfoPlus.21 Administration Help

Aspen InfoPlus.21 *Administrator Help*

Aspen InfoPlus.21 *Definition Editor Help*

DBC.21 User's Manual

Technical Support

AspenTech customers with a valid license and software maintenance agreement can register to access the online AspenTech Support Center at:

http://support.aspentech.com

This Web support site allows you to:

- Access current product documentation
- Search for tech tips, solutions and frequently asked questions (FAQs)
- Search for and download application examples
- Search for and download service packs and product updates
- Submit and track technical issues
- Send suggestions
- Report product defects
- Review lists of known deficiencies and defects

Registered users can also subscribe to our Technical Support e-Bulletins. These e-Bulletins are used to alert users to important technical support information such as:

- Technical advisories
- Product updates and releases

Customer support is also available by phone, fax, and email. The most up-to-date contact information is available at the AspenTech Support Center at **http://support.aspentech.com**

2 Installation Setup

This chapter provides information on installing the API and linking to the API.

Installing the API Server

The API server, which is a standard component of Aspen InfoPlus.21, is distributed on the same CD-ROM as Aspen InfoPlus.21. Whenever Aspen InfoPlus.21 is installed, the API Server is installed, and whenever Aspen InfoPlus.21 starts, the API Server is started.

Installing the API

The API is available for the following operating systems:

- **Microsoft Windows 2000**

The Aspen InfoPlus.21 API is distributed on the same CD-ROM as Aspen InfoPlus.21. The API is a standard component of Aspen InfoPlus.21 and is installed whenever Aspen InfoPlus.21 is installed.

Linking to the API

A program on Windows 2000 must include "**infoplus21_api.h**" and link to the **infoplus21_api.lib** in the **%INFOPLUS21_BASE%\shared\inc** and **%INFOPLUS21_BASE%\shared\bin** directories. Sample client programs and makefiles are included in the

%INFOPLUS21_BASE%\shared\samples\infoplus21_api directory. To compile the sample program **test_name2recid.c**, type the following at the DOS prompt:

```
comp_sample name2recid.c
```

This command will produce the **name2recid.exe** executable.

Environment Variables

An environment variable (or logical) named **SETCIMRPC** should point at **SETCIMRPC.CFG**, a configuration file described in "The Configuration File" below. If **SETCIMRPC** is not defined, the client program will assume that a file named **SETCIMRPC.CFG** is in the current directory.

To enable the error and status logging for a client program, define a **SETCIMRPC_LOG** environment variable that points to a file (i.e., **SETCIMRPC.LOG**).

The Configuration File

The configuration file, which is read by the client program, contains connection information and other client and server parameters for one or more database nodes. This configuration file is read when the client program calls **INISETC()** or **DaInitialize(TRUE)**.

The configuration file is specified by the **SETCIMRPC** environment variable. If the **SETCIMRPC** environment variable is not defined, the client program assumes that a file named **SETCIMRPC.CFG** is in the current directory. If a configuration file is not found, then **INISETC()** attempts to access a local database.

The configuration file consists of one or more lines in a text file. The first line in the configuration file is the default node.

Note: For access routines such as the **NAME2RECID** routine that searches all nodes to find data, the nodes are searched in the order listed in the configuration file.

Configuration Line Syntax

Syntax for configuration file lines is as follows:

```
hostname group_number [options]
```

where *hostname* is the name of machine and *group_number* is the group number for the desired database. The [*options*] argument is a list of optional parameters described in the following section.

Optional Parameters

The optional parameters should be separated by a space and can be placed in any order. If parameters are not given, the default values are used. The following is a list of optional parameters:

- **ALIAS=alias_name**

  The *alias_name* is a string, up to 128 characters in length, that specifies an alternate name for the database node. The *alias_name* is not case sensitive and is converted to uppercase when read. The *alias_name* can be used in some functions to access node information.

- **/FATAL**

  If the connection to this node fails then **INISETC()** drops all connections and returns **0**. If the parameter is not specified, then a failed connection is ignored by **INISETC()** unless all node connections fail.
- **/I=ss**

  The retry interval in seconds (default is 60 seconds). This parameter is the minimum time allowed between reconnection attempts.
- **/NEWLOG**

  Creates a new log file and erase the old log file when the client program is executed. On VAX/VMS systems, **/NEWLOG** creates a new version unless a version number is specified in the log filename.
- **/NOERR**
- **/NOLOG**
- **/NOMSG**

  The **/NOERR, /NOLOG,** and **/NOMSG** switches set logging levels for the client program. Only one switch is needed per line. The **/NOLOG** switch turns off all logging to a log file. The **/NOERR** switch turns off all logging of error messages to a log file. The **/NOMSG** switch turns off all logging of status messages to a log file. Only two types of messages, error messages and status messages, are logged. If a **SETCIMRPC_LOG** environment variable logical is not defined, then no logging is performed. Log messages have the following format:

```
client_pid_number   date_time_stamp   nodeid   log_message
```

  where *client_pid_number* is the process identifier for the client program, *date_time_stamp* is the date and time of message, *nodeid* is the node identifier number for the connection, and *log_message* is the error or status message.
- **/P=pp**

  Specifies a server protocol number, where *pp* is a number in the range of 33,554,432 through 1,073,741,823. This parameter should only be specified for servers that do not use the default protocol number of 300,363.
- **/R=xx**

  The maximum number of retries for a reconnect before ceasing any reconnect attempts, where *xx* is the maximum number of retries (default is 3). After the maximum number of retries has been attempted, **INISETC()** must be called to reconnect and clear the retry count.
- **/S=ss (applies to UNIX servers only)**

  The maximum server idle time in seconds (default is 5 minutes). If no requests are made to a connected server before the idle time is reached, the server will exit.
- **/STDERR**

  Sends error messages to standard error. This parameter causes communication errors to be sent to standard error.
- **/T=ss**

The maximum timeout value for any RPC call to wait for a reply, where *ss* is the number of seconds to wait (default is 60 seconds).

Examples of Configuration Lines

The following are examples of configuration lines:

abox 123 /T=60 /R=3 /I=60 /STDERR

```
bbox 321 /P=777777778 /NEWLOG  /FATAL
cbox 204 /ALIAS=BOX204 /STDERR /S=900
```

3 External Tasks

This chapter describes Aspen InfoPlus.21 external tasks.

Aspen InfoPlus.21 External Tasks

External task functions are routines that are called by programs that run as database external tasks. A database external task is a detached process that responds to record activations. In other words, a database external task is a "record activation handler". Such a process must reside on the same computer as the database.

Activating records

Records can be activated in a variety of ways:

- Aspen InfoPlus.21 activates a record containing a scheduling timestamp field at the specified time.
- Aspen InfoPlus.21 can also activate a record containing a change-of-state field or a field with activation criteria when the conditions specified in the record's definition record are met.

Note: For more information on how to define records with trigger fields, see the *Aspen InfoPlus.21 Developer's Manual* and the *Aspen InfoPlus.21 Definition Editor Help File*.

- **TSK_PLAN**, an Aspen InfoPlus.21 external task, activates records when processing other records defined by **ScheduledActDef** or **COSActDef**.

Note: These records are described in the *Aspen InfoPlus.21 Database User's Manual*.

- User-written programs can also activate records directly by calling **ACTEXTSK**, an Aspen InfoPlus.21 access routine.

Starting and Stopping Tasks

Aspen InfoPlus.21 external tasks are normally started with Aspen InfoPlus.21. The Aspen InfoPlus.21 Manager allows new tasks to be added to the start list. Aspen InfoPlus.21 external tasks are stopped when Aspen InfoPlus.21 stops.

An Aspen InfoPlus.21 external task can be requested to stop using an Aspen InfoPlus.21 utility program named **STOPTSK**. The task is allowed to finish current record processing.

FORCEX, another utility program, provides an alternative to **STOPTSK**. However, call **FORCEX** only if a user-written program appears to be in a loop or is ignoring error codes returned by **EXTASKCHK**, an Aspen InfoPlus.21 access routine. **FORCEX** gets an exclusive lock on the database before forcing the task to exit. Getting the lock first ensures that an Aspen InfoPlus.21 access routine is not interrupted.

External Task Program Structure

Programs that run as Aspen InfoPlus.21 external tasks include the setcim.h include file and are linked to setcim.lib. An external task program must also call the following routines:

Routine	Description
INISETC	Initializes a connection to the Aspen InfoPlus.21 database. Returns TRUE if Aspen InfoPlus.21 is up and FALSE if Aspen InfoPlus.21 is down or the user is missing READ permission to the Aspen InfoPlus.21 database. Must be called by all programs that intend to call Aspen InfoPlus.21 database access routines, even programs such as interactive utility programs, which are not run as Aspen InfoPlus.21 external tasks.
EXTSKINI	Returns the ID of a record in the Aspen InfoPlus.21 database with the same name as the Aspen InfoPlus.21 external task. Sets up an internal mechanism that allows Aspen InfoPlus.21 to awaken the task.
EXTASKCHK	Returns the ID of the next activated record to be processed. Provides other information that assists the task in determining its function. The task exits if the returned error code is not equal to **SUCCESS**, a constant in the Aspen InfoPlus.21 header file. The error code equals **STOPTSK** if the task has been requested to exit.
EXTSKWAI	Causes the task to go to sleep. The task is awakened in response to record activations or IO.
EXTSKEND	Called by external task to undo **EXTSKINI()**.
EXTSKTIM	Called by an external task to set up a timer.
NAMEPROCESS	Renames process so **EXTSKINI** call connects to that external task record.

Sample C Program

A typical external task has the high-level form:

```
#include <setcim.h>          /*  structure and #define declarations */

Unsigned int main ( int argc, char **argv)
{
   long        task_rec_id,   /  external task record ID        *
               act_rec_id,    *  activated record ID            /
               act_ft;        /  field tag of activation field  *
                              *                                 /
                              /                                 *
                              *                                 /

   ERRBLOCK    Error          /  Aspen InfoPlus.21 errors       *
   ERRARRAY    error_message  *  from function calls            /
   short       ge;            /  string for Aspen               *
               error_length   *  InfoPlus.21 error             /
               ,              /  messages                       *
               priority,      *  length of error message        /
               code;          /  record activation priority     *
                              *  activation code from           /
                              /  record def.                    *
                              *                                 /

   char        Costail;       /  TRUE if normal activation
                              *  is following group of COS      *
                                 activations.                   /


   / *    Initialize process to access Aspen InfoPlus.21 */
   If (!INISETC())
            exit(1);                    /* Aspen InfoPlus.21 is
           down or access is denied     */

   EXTSKINI( &task_rec_id, &error );

   / *  Keep processing records until error */
   while ( error.ERRCODE == SUCCESS )
   {
     EXTASKCHK ( & act_rec_id, &act_ft, &priority,
                       &code, &costail, &error );        /*
     continue checking for record

     activations                    */
      if ( !act_rec_id )
           EXTSKWAI();
      Else
```

```
        switch(code)
        {
           case 1: do_case_1( act_rec_id, act_ft ); break;
           case 2: do_case_2( act_rec_id, act_ft ); break;
           case 3: do_case_3( act_rec_id, act_ft ); break;
          /*  and more */
        }
   }
  ERRMESS( &error,error_message,&error_length )
  printf(‘‘******EXITING******\m%
.*s\n",error_length,error_message );
  ENDSETC();
}
```

External Tasks Descriptions

External task functions include the following:

EXTASKCHK

Checks for activation requests to an external task. Also returns an error code `STOPTSK` if a request for graceful shutdown has been made. *This routine replaces the obsolete routine* **`EXTSKCHK`**.

Format

```
EXTASKCHK (actid, actft, actpri, actcod, costail, error)
```

Arguments

actid

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*actid* is the record ID of an activated record. If a 0 is returned, there are no activation requests.

actft

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*actft* is the field tag of a field in an activated record that caused the activation. If a 0 is returned, the record as a whole was activated or there are no more fields in the active ID’s record that caused the activation.

actpri

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*actpri* is the activation priority of the record.

actcod

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*actcod* is the activation from the activated record's definition record. This code can be used by the external task to determine type of processing.

costail

Data Type	Byte
Access	Output only
Mechanism	Passed by reference
Description	*costail* will be nonzero if the activation is the normal activation generated by (and following) a group of change-of-state activations for the record.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
long      actid,        /*  Activating record ID
      */
          actft;        /*  Activating field tag if any
      */
short     actprior,     /*  Activation priority            */
          actcode;      /*  Activation code
      */
char      costail;            /*  Flag: normal activation after
COS activations   */
ERRBLOCK err;
```

EXTASKCHK (&actid, &actft, &actprior, &actcode, &costail, &err);

Sample FORTRAN Program

```
INTEGER*4   ACTID
INTEGER*4   ACTFT
INTEGER*2   ACTPRIOR
INTEGER*2   ACTCODE
BYTE        COSTAIL
RECORD            /ERRBLOCK/ERR
CALL  EXTASKCHK (ACTID, ACTFT, ACTPRIOR, ACTCODE,COSTAIL, ERR)
```

EXTSKEND

Ends the interface to an external task. Negates the effect of **EXTSKINI**.

Format

```
EXTSKEND()
```

Arguments

None

Sample C Program

```
EXTSKEND();
```

Sample FORTRAN Program

```
CALL EXTSKEND()
```

EXTSKINI

Initializes the interface to an external task. The routine must be called before **EXTSKCHK**.

Format

```
EXTSKINI(tskrecid, error)
```

Arguments

tskrecid

Data Type	long word

Access	Output only.
Mechanism	Passed by reference
Description	*Tskrecid* is the ID of the task record. Some external tasks use *tskrecid* to retrieve information specific to the application. By convention, the task record and the initializing process have the same name.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an *error* code as defined in the **setcim.h** include file.

Sample C Program

```
long        tskrecid;         /*  ID of the external task record
*/
ERRBLOCK    err;
EXTSKINI (&tskrecid, &err);
```

Sample FORTRAN Program

```
INTEGER*4   TSKRECID
RECORD      /ERRBLOCK/ERR
CALL EXTSKINI (TSKRECID, ERR)
```

EXTSKTIM

Sets up a timer for an Aspen InfoPlus.21 external task.

Returns

long word

Returns 0 if a timer was canceled or was already started, or 1 if the timer was successfully started.

Format

```
EXTSKTIM(millisec)
```

Arguments

millisec

Data Type	long word
Access	Input only
Mechanism	Passed by value

Description	If no timer currently exists and *millisec* > 0, **EXTSKTIM** sets up a timer of that length, which will be detected in **SEMAPWAI** upon expiration. If *millisec* ≤ 0, **EXTSKTIM** cancels any timer currently running.

Sample C Program

```
long  status,
      millisec;
status = EXTSKTIM(millisec);
```

Sample FORTRAN Program

```
INTEGER*4   STATUS
INTEGER*4   MILLISEC

STATUS = EXTSKTIM (%VAL(MILLISEC))
```

EXTSKWAI

Waits for outstanding events, typically record activations or I/O.

Format

```
EXTSKWAI()
```

Arguments

None

Sample C Program

```
EXTSKWAI();
```

Sample FORTRAN Program

```
CALL EXTSKWAI()
```

NAME_PROCESS

Renames the process so the next call to **EXTASKINI** connects the process to the given external task record.

Format

```
NAME_PROCESS (task_name, numchrs, error)
```

Arguments

task_name

Data Type	char array
Access	Input only
Mechanism	Passed by reference
Description	*task_name* is a character pointer to the task name string.

numchrs

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* is the number of characters in the task name string.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned Aspen InfoPlus.21 error code.

Sample C Program

```
ERRBLOCK    error       /*  InfoPlus.21 error return structure
      */
short       numchars;   /*  Number of characters in task name
      */
char        *task_name = "TSK_TEST";            /*  task name
string      */
```

```
 numchars = strlen (task_name);
NAME_PROCESS (tsk_name, numchars, &error);
```

Sample FORTRAN Program

```
RECORD            /ERRBLOCK/ERROR
INTEGER*2   NUMCHARS
DATA        NUMCHARS /8/
CHARACTER*24      TASK_NAME
DATA        TASK_NAME/‘TSK_TEST’/
CALL NAME_PROCESS (%REF(TASK_NAME), %VAL (NUMCHARS), ERROR)
```

4 Access Routines

This chapter describes subroutines that access the Aspen InfoPlus.21 database. Each description provides enough information to determine what routines are available and which routine should be used for a given application.

These functions are available in two different libraries: **setcim.dll** and **infoplus21_api.dll**. Programs that call functions in **setcim.dll** must include the **setcim.h** header file, and can only access a local database. Programs that call functions in **infoplus21_api.dll** must include the **infoplus21_api.h** header file, and can access either a local or remote database.

setcim.dll links to and requires the following libraries:

- atsecuritybase.dll
- cimntutil.dll
- cimwin32util.dll
- h21sys.dll
- h21user.dll
- histconfigsys.dll
- ip21license.dll
- libc21.dll

infoplus21_api.dll links to and requires the following libraries:

- cimsrvapi.dll
- cimwin32util.dll
- ip21license.dll

Summary of Routines

Database Read Routines

D2ASCIIDB

DATA2ASCII

Converts a real value to ASCII in the format of a database field.

Converts a value to ASCII in the format of the specified format record.

DB2CHBF

Reads a data buffer from the database.

DB2DUBL

Reads a double precision real number from the database (data type = DTYPDUBL).

DB2IDFT

DB2IDFT reads a record ID and field tag from the database (data type = DTYPIDFT).

DB2LONG

Reads a long word integer from the database (data type = DTYPLONG).

DB2REAL

Reads a single precision real number from the database (data type = DTYPREAL).

DB2REID

Reads a record ID from the database (data type = DTYPREID).

DB2SHRT

Reads a short word integer from the database (data type = DTYPSHRT).

DB2XTIM

Reads an extended timestamp from the database (data type = DTYPXTIM).

MRDBOCCS

Reads multiple occurrences of multiple fields from a repeat area in one database record.

MRDBVALS

Reads multiple fields from multiple records in the database.

RDBASCII

Reads from a database field and converts it to ASCII. The ASCII conversion matches that defined by the fields, FIELD_DATATYPE and FIELD_FORMAT_RECORD (defined in its definition record).

RDBOCCS

Reads multiple occurrences of a single field from one record in the database.

RDBVALS

Reads multiple values from one record in the database.

Database Write Routines

CHBF2DB

Writes a data buffer to the database.

DUBLADD2DB

Writes a double precision real number to the database (data type = DTYPDUBL). This routine was designed to accommodate the FORTRAN calling convention.

IDFT2DB

Writes a record ID and field tag to the database (data type = DTYPIDFT).

LONG2DB

Writes a long integer to the database (data type = DTYPLONG).

MWDBOCCS

Writes multiple occurrences of multiple fields to a repeat area in one database record.

MWDBVALS

Writes multiple records in the database.

REALADD2DB

Writes a real number to the database (data type = DTYPREAL). This routine is designed to accommodate FORTRAN calling conventions.

REID2DB

Writes a record ID to the database (data type = DTYPIDFT).

SHRT2DB

Writes a short integer to the database (data type = DTYPSHRT).

WDBASCII

Converts ASCII input and writes it to a database field. The appropriate conversion is done to match the FIELD_DATA_TYPE and FIELD_FORMAT_RECORD of the target field (as defined in the definition record of *recid*).

WDBOCCS

Writes multiple occurrences of a single field to one record in the database.

WDBVALS

Writes multiple values to one record in the database.

XTIM2DB

Repeat Area Management

Writes an extended timestamp to the database (data type = DTYPXTIM).

DELOCCS

Deletes multiple occurrences from a repeat area.

INSOCCS

Inserts new occurrences in a repeat area.

MRDBOCCS

Reads values from multiple occurrences of multiple fields from a repeat area in a single database record.

MWDBOCCS

Writes values from multiple occurrences of multiple fields to a repeat area in a single database record.

RDBOCCS

Reads multiple occurrences of a field from the repeat area of one record.

RHIS21DATA

Reads multiple occurrences of multiple historical fields, reading the occurrences in "reverse" order (older occurrences first), and storing the values read into multiple data arrays. Accepts extended micro second timestamp as start and end time.

WDBOCCS

Writes multiple occurrences of a single field to one record in the database.

WHIS21DAT

Writes (or inserts) one occurrence of multiple historical fields for a given extended microsecond timestamp.

Record Manipulation

ACTRECS

Activates multiple records.

COPYREC

Copies an existing record to a record with a new record name and ID. No changes-of-state in the new record is detected as a result of the copy.

CREATEREC

Creates a new record.

DELETREC

Deletes a record. The record must be in the unusable state.

MAKUNUSA

Makes a record unusable. The record cannot be referenced by other records.

MAKUSABL

Makes a record usable.

VALIDUSA

Checks if a record is in a usable state.

Folder Records

FOLDERIN

Inserts records into a folder.

FOLDEROUT

Removes records from a folder.

ROOTFOLDER

Returns the ID of the root folder.

Definition Records

DEFINID

Returns the definition record of a given record.

GETRECLIST

Returns a list of record IDs and names which meet specific criteria.

Record and Field Information

CHKFREE

Checks if a record ID is available for use.

DECODFT

Decodes a field name to its field tag.

DECODNAM

Decodes a record name to its record ID.

DECODRAF

Decodes a record name and associated field name to their respective record ID and field tag.

FIELDDEFNINFO

Returns database information about a field

FIELDINFO

Returns information about a field.

FTNAME2FT

Converts a field name to a field tag. FTNAM2FT returns 0 if the field name is invalid.

GETFTDB

Returns the field name given the record ID and field tag.

GETNAMDB

Returns the record name given the record ID.

GETNMFDB

Returns the record and field names given to a record ID and field tag.

GETRECLIST

Returns a list of record IDs and names which meet specific criteria.

NAMSZDEF

Returns the record name size from a definition record.

RECTYPEOK

Returns TRUE if the specified record is of a given type.

VALIDREC

Checks whether a record ID is used.

VALIDUSA

Checks if a record is in a usable state.

Security Routines

GETDBPERMIS

Verifies a user's database permissions specified in the *wantedDbPermis* argument.

GETFLDPERMIS

Verifies a user's write permissions against a field in a record.

GETRECPERMIS

Verifies a user's record permissions specified in the *wantedRecPermis* argument.

GETWRITELEVEL

Gets the write-level permission value of the specified field in the record. Write level is used in coordination with the record *write* permissions to restrict a user's ability to modify a field. To read a field's write level, the user must have a read permission to the record.

History Routines

RHIS21AGGREG

Generates statistics for each time interval between two specified times.

RHIS21DATA

Reads multiple occurrences of multiple historical fields, reading the occurrences in chronological order (older occurrences first), and storing the values read into multiple data arrays. Requires archiving to be on. Accepts extended micro second timestamp as start and end time.

RHIS21REV

Reads multiple occurrences of multiple historical fields, reading the occurrences in reverse chronological order (newer occurrences first), and storing the values read into multiple data arrays. Will read memory occurrences when archiving is off or for history defined without archiving. Accepts extended micro second timestamp as start and end time.

HISOLDESTOK

Obtains and/or changes the oldest allowed time for the history repeat area of a record.

Note: A utility program, xoldestok.exe, can also be used to change the oldest allowed time.

Converting from ASCII

WHIS21DAT

Writes (or inserts) one occurrence of multiple historical fields for a given extended microsecond timestamp.

ASCII2XTS

Converts ASCII time to an extended timestamp.

ASCIIDB2I

Converts ASCII in the format of a database field to an integer.

Converting to ASCII

D2ASCIIDB

Converts a real value to ASCII in the format specified by a database field.

DATA2ASCII

Converts a value to ASCII in the format of the specified format record.

I2ASCIIDB

Converts an integer to ASCII in the format specified by a database field.

R2ASCIIDB

Converts a real value to ASCII in the format specified by a database field.

TMST2ASCII

Converts a timestamp in to the current Aspen InfoPlus.21 ASCII date/time format.

XTS2ASCII

Converts an extended timestamp into the current Aspen InfoPlus.21 ASCII data/time format.

External Task

ACTRECS

Activates multiple records for processing by external tasks.

Error, Summary, and Log Routines

ERRMESS

Returns ASCII message text for database error block.

LOGMESS

Adds a message to a log record.

Database Parameter Routines

RECIDMAX

Obtains the highest record ID in the database as defined in the ENGCON utilities menu.

Timestamp Manipulation

DB2XTIM

Reads an extended timestamp from the database.

DSPDT2XTS

Converts "day of century" time format to an Aspen InfoPlus.21 extended timestamp.

GETDBXTIM

Returns the current database time as an Aspen InfoPlus.21 extended timestamp.

TIME2DB

Writes a timestamp to the database.

TS2XTS

Converts an Aspen InfoPlus.21 timestamp to the equivalent extended timestamp. The **XTSFAST** element of the extended timestamp is the same as the original timestamp.

XTIM2DB

Writes an extended timestamp to the database.

XTS2ASCII

Converts an extended timestamp to Aspen InfoPlus.21 ASCII date/time format.

XTS2DSPDT

Converts an Aspen InfoPlus.21 extended timestamp to the day of the century and the time. **XT2DSPDT** automatically incorporates the Aspen InfoPlus.21 system time offset into the conversion.

Miscellaneous Routines

DELOCCS

Deletes a number of occurrences from a record.

ERRMESS

Returns the ASCII error message corresponding to a database error block.

INSOCCS

Inserts a number of occurrences in the given record.

NXTREFER

Finds the next reference to a given record and optionally to a given field.

RESTORSNAP

Reads a database snapshot disk file into memory.

SAVESNAP

Writes a database snapshot to a disk file.

Initialization and Completion

ENDSETC

Ends support in a program (operating system dependent).

INISETC

Starts Aspen InfoPlus.21 support in a program (operating system dependent). This routine must be called before any Aspen InfoPlus.21 access routines are called.

Access Routine Abstracts

The routines in this chapter give programs access to the Aspen InfoPlus.21 database. They provide an interface between Aspen InfoPlus.21 and external tasks or other programs that interact with the database. These routines:

- interface to external tasks
- read data from records in the database
- write data to records in the database
- return information about records and fields
- convert record and field names to and from IDs
- translate values to and from ASCII strings
- create and delete records
- change record usability
- set various database parameters
- interface to history
- generate log and summary entries
- find the next or last value for database entities
- support special real numbers (NaN, +/- infinity)
- change fields in a definition record that are normally unchangeable

Connect and Disconnect

When a program linked to s**etcim.dll** calls **INISETC()**, the **INISETC()** function maps the program into the local database's shared memory. **INSETC()** returns 1 if the local database is up and 0 if the local database is down.

When a program linked to **setcim.dll** calls **ENDSETC()**, the **ENDSETC()** function unmaps the program from the local database's shared memory.

A program linked to **infoplus21_api.dll** calls **INISETC()** to connect to one or more nodes. **INISETC()** reads the configuration file, builds the internal node list, and connects to all nodes. It returns a 0 if all connections failed or if a FATAL connection failed. The function returns a 1 if at least one connection was successful.

A program linked to **infoplus21_api.dll** should call **ENDSETC()** before exiting. **ENDSETC()** disconnects all Server Nodes, destroys the Server List Functions, and releases all the resources. No program should call **ENDSETC()** except when exiting.

Record Names and IDs

Most of the database access functions accept record IDs as arguments. A record ID is a 32-bit integer that identifies the record to be accessed. The least significant 16 bits contains the record ID. A Server Index may be embedded in some bits in the most significant bits of the record ID. The Server Index is an index into the API's server node list.

The function **DECODNAM()** accepts a string containing a record name and returns a long integer containing the corresponding record ID.

When a program linked to **setcim.dll** calls **DECODNAM()**, any record ID returned will reside in the local database.

When a program linked to **infoplus21_api.dll** calls **DECODNAM()**, then **DECODNAM()** searches all connected databases for the record with the given name. If it finds a database containing the specified record, it will return a long record ID. A portion of the 16 most significant bits in the returned record ID will contain the node ID. The 16 least significant bits of the returned record ID will contain the ID of the record within that database.

The function **DECODRAF()** works in a similar fashion as **DECODNAM()** except that it returns a record ID and field tag after accepting a string containing a record name and field name.

After **DECODNAM()** or **DECODRAF()** returns a record ID, the calling program can then pass the returned record ID as input to other database access functions.

Some database access functions accept multiple record IDs as inputs. Generally, these functions require all record IDs to have the same embedded node ID. If any Server Index does not match, the routine returns a NODE_MISMATCH error.

ACTRECS

Activates multiple records for processing by external tasks.

Format

```
ACTRECS(numrec, recids, actprio, oldprio, numact, error)
```

Arguments

numrec

Data Type	long word
Access	Input Only
Mechanism	Passed by value
Description	*numrec* is the number of records (contained in the recids array) that are to be activated.

recids

Data Type	long word array
Access	Input Only
Mechanism	Passed by reference
Description	*recids* is an array of record IDs to be activated.

actprio

Data Type	short word
Access	Input Only
Mechanism	Passed by value
Description	*actprio* is the activation priority with which all records are activated.

oldprio

Data Type	short word array
Access	Output only
Mechanism	Passed by reference
Description	*oldprio* is optionally an array of old activation priorities. The previous activation priority is returned for each successfully activated record. If the record was not previously activated, 0 is returned. If *oldprio* is NULL, then no previous priorities are returned.

numact

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description:	*numact* is the number of records that were successfully activated.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* returns an error code as defined in the **setcim.h** include file.

Sample C Program

```
Long        numrec = 3;     /*  number in array          */
Long        recids[3];      /*  Record IDs to activate   */
Short       actprio,        /*  activation priority      */
            oldprio[3];     /*  old priority             */
Long        numact;         /*  number activated         */
ERRBLOCK    error;          /*  InfoPlus.21 error block  */

recids[0] = 1500;
recids[1] = 1600;
recids[2] = 1700;
Actprio = 12;

ACTRECS (numrec, recids, actprio, oldprio, &numact, &error);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*2       ACTPRIOR
INTEGER *4      NUMREC
INTEGER *4      RECIDS(3)
INTEGER *2      ACTPRIO
INTEGER *2      OLDPRIO(3)
INTEGER *4      NUMACT
RECORD          /ERRBLOCK/ERROR


NUMREC = 3
ACTPRIO =
12
RECIDS(1) =
1500
RECIDS(2) =
1600
RECIDS(3) =
1700


CALL  ACTRECS
(%VAL(NUMERIC),RECIDS,%VAL(ACTPRIO),OLDPRIO,NUMACT,ERROR)
```

ASCII2XTS

Converts ASCII time to an extended timestamp.

Format

```
ASCII2XTS(ptbuff, sizebuff, xts, error)
```

Arguments

ptbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* specifies the address of the buffer containing the ASCII data.

sizebuff

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*sizebuff* specifies the number of characters in the buffer.

xts

Data Type	XTSBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*xts* is the converted extended timestamp.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*error* = 0 if no error. Otherwise *error* = 1.

Sample C Program

```
Char      ptbuff[20];   /* Address of the buffer containing the ASCII data   */
Short     sizebuff;     /* Number of characters in the buffer                */
Char      error;        /* Returns TRUE if the char are in an invalid format */
XTSBLOCK  xts;          /* Extended timestamp                                */
```

ASCII2XTS (ptbuff, sizebuff, &xts, &error);

Sample FORTRAN Program

```
CHARACTER*20    PTBUFF
INTEGER*2       SIZEBUFF
INTEGER*4       TIME_STAMP
BYTE            ERROR

CALL ASCII2XTS (%REF(PTBUFF), %VAL(SIZEBUFF), XTS, ERROR)
```

ASCIIDB2I

Converts an ASCII string to an integer value. The conversion is based on the format of the integer field specified. If the field is formatted as a delta time, the string must contain a valid delta time.

Format

```
ASCIIDB2I (recid, ft, ptbuff, numchars, indata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the database field to be converted.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the integer field identifying the format.

ptbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer containing the ASCII data.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* identifies the number of characters in the buffer.

indata

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*indata* is the result of the ASCII to integer conversion.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
Long        recid,      /* Record ID                          */
            ft,         /* Field tag                          */
            indata;     /* Returned integer value             */
Char        ptbuff[20]; /* Buffer containing the data         */
Short       numchars;   /* Number of characters in the buffer */
ERRBLOCK    err;        /* Error return status                */

ASCIIDB2I (recid, ft, ptbuff, numchars, &indata,
&err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       FT
INTEGER*4       INDATA
CHARACTER*20    PTBUFF
INTEGER*2       NUMCHARS
RECORD          /ERRBLOCK/ERR

CALL ASCIIDBI ( %VAL(RECID), %VAL(FT), %REF(PTBUFF),
%VAL(NUMCHARS),
```

INDATA, ERR)

CHBF2DB

Writes characters or a data buffer to a database field.

Format

```
CHBF2DB(recid, ft, ptdbfr, numbytes, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* specifies the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* specifies the field tag of the field to contain the data.

ptbfr

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the source buffer.

numbytes

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numbytes* specifies the number of bytes in the buffer. For character fields, if *numbytes* = 0, then the field will be filled with blanks. For scratch pad fields, *numbytes* must equal the field length.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
Long        recid,          /* The record ID of the record to contain the data   */
            ft,             /* The field tag of the field to contain the data    */
Char        ptdbfr[600];    /* Address of the source buffer                      */
Char        ptbuff[20];     /* Buffer containing the data                        */
Short       numbytes;       /* Number of bytes in the buffer                     */
ERRBLOCK    err;            /* Error return status                               */

CHBF2DB (recid, ft, ptdbfr, numbytes, &err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       FT
BYTE            PTDBFR(600)
CHARACTER*20    PTBUFF
INTEGER*2       NUMBYTES
RECORD          /ERRBLOCK/ERR

CALL CHBF2DB ( %VAL(RECID), %VAL(FT), PTDBFR,
%VAL(NUMBYTES), ERR)
```

CHKFREE

CHKFREE checks if a record ID is available for use.

Format

```
CHKFREE(freeid, error)
```

Arguments

freeid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*freeid* specifies the record ID to be tested.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        freeid;     /*  Record ID to      */
                        be tested

ERRBLOCK    err;


CHKFREE (freeid, &err);
```

Sample FORTRAN Program

```
INTEGER*4   FREEID

RECORD      /ERRBLOCK/ERR


CALL CHKFREE ( %VAL(FREEID), ERR)
```

CHKFTREC

Checks if a record has a given field.

Format

```
CHKFTREC(ftcheck, recid, error)
```

Arguments

ftcheck

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ftcheck* is the field tag for the field of interest.

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* specifies the record ID of the record being examined.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,    /* Record ID of the record being examined    */
            ftcheck   /* Field tag for the field of interest       */
            ;
ERRBLOCK    err;

CHKFTREC (ftcheck, recid,
&err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       FTCHECK
RECORD          /ERRBLOCK/ERR

CALL CHKFTREC ( %VAL(FTCHECK), %VAL(RECID), ERR)
```

COPYREC

Copies an existing record to a record with a new record name and ID. No changes-of-state in the new record will be detected as a result of the copy.

Format

```
COPYREC(recid, newid, substids, ptname, numchars, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*newid* is the record ID of the record to be copied.

newid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*newid* specifies the record ID to be assigned to the new record.

substids

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*substids* is nonzero if record or record and field references to *recid* should be replaced with references to *newid* in the new record. A zero indicates references should not be changed.

ptname

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptname* contains the address of a buffer containing the new record's name.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of characters in the buffer.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /*  Record ID of the record to copy      */
            newid;      /*  Record ID to be assigned to the      */
                        new record
char        ptname[3    /*  Address of  buffer containing        */
            0];         the new record's        name
ERRBLOC     err;
K

COPYREC (recid, newid, 1, ptname, numchars,
&err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       NEWID
CHARACTER*      PTNAME
30
INTEGER*2       NUMCHARS
RECORD          /ERRBLOCK/ERR

CALL COPYREC( %VAL(RECID), %VAL(NEWID), %VAL(1),
%REF(PTNAME), %VAL(NUMCHARS), ERR)
```

CREATEREC

Creates a new record using the indicated definition record and ID.

Format

```
CREATEREC(recid, defid, ptname, numchars, err)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* specifies the record ID to be assigned to the new record (must not be an ID used by another record).

defid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*defid* specifies the record ID of the definition record for *recid*.

ptname

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptname* contains the address of a buffer containing the new record's name.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by reference
Description	*numchars* specifies the number of characters in the buffer.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*err* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        Recid,      /* Record ID to be assigned to the new record */
            Defid;      /* Record ID of the definition record for recid */
char        Ptname[30]; /*  Address of a buffer containing the new record's name */
short       Numchars;   /* Number of characters in the buffer */
ERRBLOCK    err;

CREATEREC (recid, defid, ptname, numchars,
&err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       DEFID
CHARACTER*30    PTNAME
INTEGER*2       NUMCHARS
RECORD          /ERRBLOCK/ERR

CALL CREATEREC ( %VAL(RECID), %VAL(DEFID),
%REF(PTNAME), %VAL(NUMCHARS), ERR)
```

D2ASCIIDB

Converts a real value to ASCII in the format specified by a database field.

Format

```
D2ASCIIDB(recid, ft, realdata, ptbuff, maxchars, numchars, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*Recid* is the record ID for the field tag.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*Ft* is the field tag of the field defining the format.

realdata

Data Type	Double precision real
Access	Input only
Mechanism	Passed by reference
Description	*Realdata* is the real value to be converted.

ptbuff

Data Type	character array
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer to receive the ASCII data.

maxchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*maxchars* specifies the maximum number of characters in the buffer.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the number of characters that are normally written to a buffer. If *numchars* is greater than *maxchars*, only *maxchars* characters are used.

e**rror**

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /*  Record ID                          */
            ft;         /*  Field tag                          */
double      realdata;   /*  Real value                         */
char        ptbuff[100]; /*  Address of buffer to receive the ASCII data */
short       maxchars,   /*  Maximum number of characters in the buffer */
            numchars;   /*  Number of characters normally written to a buffer */
ERRBLOCK    err;


maxchars = 100;

D2ASCIIDB(recid, ft, &realdata, ptbuff, maxchars,
&numchars, &err);
```

Sample FORTRAN Program

INTEGER*4	RECID
INTEGER*4	FT
REAL*8	REALDATA
CHARACTER*100	CHARACTER*100
INTEGER*2	MAXCHARS
INTEGER*2	NUMCHARS
RECORD	/ERRBLOCK/ERR

MAXCHARS = 100

CALL D2ASCIIDB(%VAL(RECID), %VAL(FT), REALDATA, %REF(PTBUFF), %VAL(MAXCHARS), NUMCHARS, ERR)

DATA2ASCII

Converts a value to ASCII as specified by the given format record.

Format

```
DATA2ASCII(ptdata, formid, datatype, scpd_flag, ptbuff, maxchars,
numchars, error)
```

Arguments

ptdata

Data Type	Pointer to data
Access	Input only
Mechanism	Passed by reference
Description	*ptdata* is the address of the data to format. The value must be of the datatype specified by *datatype.*

formid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*formid* is either zero or the ID of a format record to use in formatting the ASCII string.

datatype

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*datatype* is the data type indicator as defines in the **setcim.h** include file.

scpd_flag

Data Type	Byte
Access	Input only
Mechanism	Passed by value
Description	*scpd_flag* is only used when *datatype* is positive. A value of zero indicates the field is to be formatted as character string data. A value of one indicates the field is to be formatted as a scratch pad field.

ptbuff

Data Type	character array
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* is the buffer to receive the ASCII data.

maxchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*maxchars* specifies the maximum number of ASCII characters in *ptbuff* to use.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the number of characters that are required to hold the data. If *numchars* is greater than *maxchars*, only *maxchars* are used. If there is no format record defined for the field, then *numchars* will always be equal to *maxchars* + 1.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        formid;     /*  Record ID of the format record      */
char        ptbuff[100];/*  ASCII data                          */
short       datatype,   /*  Data type == DTYP????               */
            maxchars,   /*  Max number of ASCII characters to use */
            numchars;   /*  Number of ASCII characters used     */
Float       data;       /*  Real value to format                */
ERRBLOCK    err;

datatype = DTYPREAL;
maxchars = 80;

DATA2ASCII (&data, formid, datatype, 0, ptbuff, maxchars, &numchars, &err);
```

Sample FORTRAN Program

```
INTEGER*4          FORMID
INTEGER*4          DATATYPE
CHARACTER*80       PTBUFF
CHARACTER*100      CHARACTER*100
INTEGER*2          MAXCHARS
INTEGER*2          NUMCHARS
REAL*4             DATA
RECORD     /ERRBLOCK/ERR

MAXCHARS = 80
DATATYPE = DTYPREAL

CALL DATA2ASCII(DATA, %VAL(FORMID),
%VAL(DATATYPE), $VAL(0), %REF(PTBUFF),
%VAL(MAXCHARS),1,NUMCHARS,ERR)
```

DB2CHBF

Reads a character or a data buffer from the database.

Format

```
DB2CHBF(recid, ft, ptdbfr, numbytes, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* specifies the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field containing the data.

ptdbfr

Data Type	character array
Access	Output only
Mechanism	Passed by reference
Description	*ptdbfr* specifies the address of the destination buffer.

numbytes

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numbytes* specifies the number of bytes in the buffer.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record containing the data */
            ft;         /* Field tag of the field containing the data */
char        ptdbfr[40];  /* Destination buffer */
short       numbytes;    /* The number of bytes in the buffer */
float       data;        /*  Real value to format */
ERRBLOCK    err;                                    /*
```

```
Numbytes = 40;
DB2CHBF(recid, ft, ptdbfr, numbytes,
&err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       FT
CHARACTER*40    PTDBFR
INTEGER*2       NUMBYTES
RECORD          /ERRBLOCK/ERR
numbytes = 40
```

```
CALL DB2CHBF( %VAL(RECID), %VAL(FT), %REF(PTDBFR),
%VAL(NUMBYTES), ERR)
```

DB2DUBL

Reads a double precision real number from the database (data type = DTYPDUBL).

Format

```
DB2DUBL(recid, ft, dubldata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* specifies the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* specifies the field tag of the field containing the data.

dubldata

Data Type	double precision
Access	Output only
Mechanism	Passed by reference
Description	*dubldata* is the double precision value of the record ID specified.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record containing the data */
            ft;         /* Field tag of the field containing the data */
double      dubldata;   /* Double precision value */
ERRBLOCK    err;

DB2DUBL (recid, ft, &dubldata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
REAL*8      DUBLDATA
RECORD      /ERRBLOCK/ERR

CALL DB2DUBL ( %VAL(RECID), %VAL(FT), DUBLDATA, ERR)
```

DB2IDFT

Reads a record ID and field tag from the database (data type = DTYPIDFT). For example, this routine could be used to get the record ID and field tag of the field that caused a change-of-state activation.

Format

```
DB2IDFT(recid, ft, idftdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* specifies the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* specifies the field tag of the field containing the data.

idftdata

Data Type	IDANDFT
Access	Output only
Mechanism	Passed by reference
Description	*idftdata* is the record ID and field tag values.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,    /*  Record ID of the record containing the data   */
            ft;       /*  Field tag of the field containing the data     */
IDANDFT     idftdata; /*  Record ID and field tag values                 */
ERRBLOCK    err;


DB2IDFT(recid, ft, &idftdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       FT
RECORD          /IDANDFT/IDFTDATA
RECORD          /ERRBLOCK/ERR

CALL DB2IDFT ( %VAL(RECID), %VAL(FT), IDFTDATA, ERR)
```

DB2LONG

Reads a long word integer from the database (data type = DTYPLONG).

Format

```
DB2LONG(recid, ft, intdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field containing the data.

intdata

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*intdata* is the result of the database read as a long word integer value.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record          */
                        containing the data
            ft;         /* Field tag of the field containing */
                        the data
            intdata;    /* long word integer value          */
ERRBLOCK    err;


DB2LONG (recid, ft, &intdata,
&err);
```

Sample FORTRAN Program

```
INTEGER*    RECID
4

INTEGER*    FT
4

INTEGER*    INTDATA
4

RECORD      /ERRBLOCK/ERR


CALL DB2LONG ( %VAL(RECID), %VAL(FT), INTDATA,
ERR)
```

DB2REAL

Reads a single precision real number from the database (data type = DTYPREAL).

Format

```
DB2REAL(recid, ft, realdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field containing the data.

realdata

Data Type	single precision real
Access	Output only
Mechanism	Passed by reference
Description	*realdata* is the single precision real value read from the database.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record          *
                        containing the data                  */
            ft;         /* Field tag of the field           *
                        containing the data                  */
Float       realdat     /* Real value                       *
            a;                                               /
ERRBLOC     err;
K
DB2REAL (recid, ft, &realdata, &err);
```

Sample FORTRAN Program

```
INTEGER*    RECID
4
INTEGER*    FT
4
REAL*4      REALDATA
RECORD      /ERRBLOCK/ERR

CALL DB2REAL (%VAL(RECID), %VAL(FT), REALDATA,
ERR)
```

DB2REID

Reads a record ID from the database (data type = DTYPREID).

Format

```
DB2REID(recid, ft, iddata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field containing the data.

iddata

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*iddata* is the record ID value read from the database.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record        */
                        containing the data
            ft;         /* Field tag of the field         */
                        containing the data
            iddata;     /* Record ID value                */
ERRBLOCK    err;

DB2REID (recid, ft, &iddata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID

INTEGER*4   FT

INTEGER*4   IDDATA

RECORD      /ERRBLOCK/ERR

CALL DB2REID ( %VAL(RECID), %VAL(FT), IDDATA,
ERR)
```

DB2SHRT

Reads a one-word integer from the database (data type = DTYPSHRT).

Format

```
DB2SHRT (recid, ft, shrtdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field containing the data.

shrtdata

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*shrtdata* is the short word integer value read from the database.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record containing the data */
            ft;         /* Field tag of the field containing the data */
short       shrtdata;   /* short word integer value */
ERRBLOCK    err;

DB2SHRT (recid, ft, &shrtdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4       RECID
INTEGER*4       FT
INTEGER*2       SHRTDATA
RECORD          /ERRBLOCK/ERR

CALL DB2SHRT ( %VAL(RECID), %VAL(FT),
SHRTDATA, ERR)
```

DB2XTIM

Reads an extended timestamp from the database (data type = DTYPXTIM).

Format

```
DB2XTIM(recid, ft, xtsdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* represents the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* represents the field tag of the field containing the data.

xtsdata

Data Type	XTSBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*xtsdata* represents the extended timestamp value read from the database.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID of the record          */
                        containing the data
            ft;         /* Field tag of the field           */
                        containing the data
ERRBLOCK    err;        /*  Error return status             */
XTSBLOCK    xtsdata;    /* Extended timestamp               */

DB2XTIM (recid, ft, &xtsdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
RECORD      /XTSBLOCK/XTSDATA
RECORD      /ERRBLOCK/ERR

CALL DB2XTIM (%VAL(RECID), %VAL(FT), XTSDATA, ERR)
```

DECODFT

Decodes a field name to its field tag.

Format

```
DECODFT(ptbuff, numchars, ft, error)
```

Arguments

pfbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer containing the field type name.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* represents the number of characters in the buffer.

ft

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*ft* is the returned field tag of the field name requested. If the field is not found, *ft* returns 0.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
char   *pbuff  =  "NAME";    /* Address of the buffer containing the field type name*/
short   numchars = 4;        /* Number of characters in the buffer       */
long    ft;                  /* Field tag of the field.  If the field is not found */
ERRBLOCK err;


DECODFT (ptbuff, numchars, &ft, &err);
```

Sample FORTRAN Program

```
CHARACTER*4        PTBUFF
DATA          PTBUFF/'NAME'/
INTEGER*2     NUMCHARS
DATA          NUMCHARS/4/
INTEGER*4     FT
RECORD              /ERRBLOCK/ERR

CALL DECODFT ( %REF(PTBUFF), %VAL(NUMCHARS), FT, ERR)
```

DECODNAM

Decodes a record name to its record ID.

Format

```
DECODNAM(ptbuff, numchars, recid, error)
```

Arguments

ptbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer containing the record name.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* is the number of characters in the record name given in *ptbuff.*

recid

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*recid* is the record ID of the record. *recid* returns 0 if the name is all blanks and -1 if the name is not found.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
char    *ptbuff = "DefinitionDef";     /*  Record name              */
long    recid;                  /* Record ID of the record    */
short   numchars = 13;                  /*  # of characters in the buffer */
ERRBLOCK err;

DECODNAM (ptbuff, numchars, &recid, &err);
```

Sample FORTRAN Program

```
CHARACTER*13            PTBUFF
DATA                PTBUFF/'DefinitionDef'/
INTEGER*4           RECID
INTEGER*2           NUMCHARS
DATA                NUMCHARS/13/
RECORD                  /ERRBLOCK/ERR

CALL DECODNAM (%REF(PTBUFF), %VAL(NUMCHARS), RECID, ERR)
```

DECODRAF

Decodes a record name and associated field name to their respective record ID and field tag.

Format

```
DECODRAF(ptbuff, numchars, recid, ft, error)
```

Arguments

ptbuff

Data Type	NAMFTARR
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the buffer containing the record and field names.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the maximum number of ASCII characters in *ptbuff* to use.

recid

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*recid* is the record ID of the record.

ft

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*ft* is the field tag of the field.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
NAMFTARR    ptbuff;    /*  Buffer containing the record and field names*/
short       numchars;  /* Number of characters in ptbuff     */
long        recid,     /*  Record ID of the record           */
            ft;        /*  Field tag of the field        */
ERRBLOCK    err;

DECODRAF (ptbuff, numchars, &recid, &ft, &err);
```

Sample FORTRAN Program

```
INTEGER*2   NUMCHARS
INTEGER*4   RECID
INTEGER*4   FT
RECORD      /NAMFTARR/PTBUFF
RECORD      /ERRBLOCK/ERR

CALL DECODRAF(%REF(PTBUFF), %VAL(NUMCHARS), RECID, FT, ERR)
```

DEFINID

Returns the definition record of a given record.

Returns

longword

Returns 0 if the record ID is invalid.

Format

```
DEFINID(recid)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

Sample C Program

```
long    recid,          /* Record ID of the record   */
        defid;          /* Definition record ID      */
defid = DEFINID(recid);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    DEFID
DEFID = DEFINID (%VAL(RECID))
```

DELETREC

Deletes a record. The record must be in the unusable state.

Format

```
DELETREC(recid, err)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid;  /* Record ID of the record   */
ERRBLOCK    err;

DELETREC (recid, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
RECORD      /ERRBLOCK/ERR

CALL DELETREC (%VAL(RECID), ERR)
```

DELOCCS

DELOCCS deletes occurrences between two existing occurrences. This changes the occurrence numbers of all occurrences from the oldest to the point of deletion. The point of deletion is specified by an occurrence number. In other words, occurrences which are before the specified occurrence number will not change, all occurrences after the deletion point will be shifted up.

Format

```
DELOCCS(recid, ft, numoccs, occnum,  occsdeleted,   error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the Record ID of the record being accessed.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is a field tag of a field in the record's nonhistorical repeat area (records can have more than one repeat area). To be a valid field the occurrence number needs to be specified, normally set to one.

numoccs

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*numoccs* is the number of occurrences to be deleted.

occnum

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*occnum* is the occurrence number with which to begin the deletion.

occsdeleted

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*occsdeleted* is the number of occurrences deleted successfully.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned Aspen InfoPlus.21 error code.

Sample C Program

```
long        recid,          /*  Record ID                    */
            ft,             /*  Field in non–history repeat area*/
            numoccs = 5,    /*  Number of occurrences to be deleted*/
            occnum;             /*  Deletion point                  */
short       occsdeleted;    /*  Returned # of occurrences deleted   */
ERRBLOCK    err;            /*  InfoPlus.21 error return structure  */

DELOCCS (recid, ft, numoccs, occnum, &occsdeleted, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
INTEGER*4    NUMOCCS
INTEGER*4    OCCNUM
INTEGER*2    OCCSDELETED
RECORD              /ERRBLOCK/ERR
NUMOCCS = 5

CALL DELOCCS( %VAL(RECID), %VAL(FT), %VAL(NUMOCCS),
%VAL(OCCNUM), OCCSDELETED, ERR)
```

DSPDT2XTS

Converts "day of century" time format to an Aspen InfoPlus.21 extended timestamp.

Format

```
DSPDT2XTS (day, time, xts)
```

Arguments

day

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*day* is the number of days into the century.

time

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*time* is the number of tenths of second into the day of the century.

xts

Data Type	XTSBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*xts* is the InfoPlus.21 extended timestamp.

Sample C Program

```
long        day;   /*  # of days into the century              */
long        time;  /*  # of 1/10 second into the day   */
XTSBLOCK    xts;   /*  InfoPlus.21 extended timestamp       */
```

Sample FORTRAN Program

DSPDT2XTS (day, time, &xts);

```
INTEGER*4   DAY
INTEGER*4   TIME
RECORD              /XTSBLOCK/XTS

CALL DSPDT2XTS(%VAL(DAY), %VAL(TIME), XTS)
```

DUBLADD2DB

Writes a double-precision real number to the database (data type = DTYPDUBL). This routine was designed to accommodate the FORTRAN calling conventions.

Format

```
DUBLADD2DB(recid, ft, dubldata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record to contain the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field to contain the data.

dubldata

Data Type	double precision real
Access	Input only
Mechanism	Passed by reference
Description	*dubldata* is the double-precision value to be stored in the database.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,              /* Record ID of the record to contain the data */
            ft;          /* Field tag of the field to contain the data      */
double      dubldata;  /* Double precision value                          */
ERRBLOCK err;

DUBLADD2DB (recid, ft, &dubldata, &err)
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
REAL*8               DUBLDATA
RECORD       /ERRBLOCK/ERR

CALL DUBLADD2DB (%VAL(RECID), %VAL(FT), DUBLDATA, ERR)
```

ENDCONS

Terminates an interface for interprocess communications between an active GCS/InfoPlus.21 console server process and the calling application program. Such an interface is initialized by calling the access routine **INITCONS**.

Format

```
ENDCONS (descriptor, error)
```

Arguments

descriptor

Data Type	Console
Access	Input and output
Mechanism	Passed by reference
Description	*descriptor* is a reference to a structure that identifies the interface that has been initialized. This structure must have been initialized by a call to **INITCONS**. On output, this structure no longer describes a valid interface. The data type Console is defined in the header file **console.h**.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* indicates whether or not the interface was successfully terminated. If *error.ERRCODE* is not set to SUCCESS, then some resources could not be deallocated or cleaned up. The structure referenced by descriptor cannot be used regardless of the value returned by *error*. The following values of *error*.ERRCODE are possible: CLFILERR ERSCGCSI NOSCGCSI RDSCGCSI WRSCGCSI

Sample C Program

```
#include <setcim.h>
#include <console.h>

long            console = CONSOLE_TASK_RECID;
Console         descriptor;
ERRARRAY  error_msg;
ERRBLOCK  error;
short           errsz;

/*  Verify that InfoPlus.21 is running */

if (INISETC ())
{
   INITCONS (console, &descriptor, &error);
   if (error.ERRCODE == SUCCESS)
   {
      <can call other GCS/InfoPlus.21 console access routines here>
      /* Terminate interface with console session */

      ENDCONS (&descriptor, &error);
   }
   /*  Handle error returned from either INITCONS or ENDCONS */

   if (error.ERRCODE != SUCCESS)
   {
       ERRMESS (&error, error_msg, &errsz);
       error_msg[errsz] = '\0';
       printf ("%s\n", error_msg);
  }
  /* Terminate interface with InfoPlus.21 */

  ENDSETC ();

}
else
  printf ("InfoPlus.21 is not up!\n");
```

Sample FORTRAN Program

```
INCLUDE     setcim.inc
INCLUDE      console.inc

INTEGER*4  console;
RECORD      /Console/  descriptor;
RECORD      /ERRARRAY/ error_msg;
RECORD      /ERRBLOCK/ error;
INTEGER*2   errsz;

DATA console /892/
```

```
C  Verify that InfoPlus.21 is up
IF (INISETC () .NE. 0) THEN
        CALL INITCONS (%VAL(console), descriptor, error)
        IF (error.ERRCODE .EQ. SUCCESS) THEN
          <can call other GCS/InfoPlus.21 console access routines here>
C  Terminate the interface with the console session
CALL ENDCONS (descriptor, error)
ENDIF
C  Handle error returned from either INITCONS or ENDCONDS
        IF (error.ERRCODE .NE. SUCCESS) THEN
          CALL ERRMESS (error, %REF(error_msg), errsz)
          PRINT 5, error_msg(1:errsz)
        ENDIF
C  Terminate interface with InfoPlus.21
        CALL ENDSETC ()
        ELSE
         PRINT *, 'InfoPlus.21 is not up'
        ENDIF

5       FORMAT (' ', A)
```

ENDSETC

Ends Aspen InfoPlus.21 support in a program (operating system dependent).

Format

```
ENDSETC()
```

Arguments

None

Sample C Program

```
main()
{
  if( !INISETC() )
  {
    /*  InfoPlus.21 is DOWN or no READ access*/
    exit():
  }
       .
       .
       .
  ENDSETC();
}
```

Sample FORTRAN Program

```
PROGRAM MYPROG

   IF( INISETC() .EQ. 0 ) THEN
C  setcim is down
   CALL  EXIT
   ENDIF
CALL  ENDSETC()
CALL  EXIT
END
```

ERRMESS

Returns the ASCII error message corresponding to a database error block.

Format

```
ERRMESS(error, error_msg, errsz)
```

Arguments

error

Data Type	ERRBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*error* is a database *error* block containing the *error* codes to be translated to ASCII.

error_msg

Data Type	ERRARRAY
Access	Output only
Mechanism	Passed by reference
Description	*error_msg* is the array containing the translated ASCII *error* message.

errsz

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*errsz* is the number of characters in the returned message. The remaining characters in *errarr* are filled with blanks.

Sample C Program

```
ERRARRAY    error_msg;   /* Error message array                */
ERRBLOCK    err;         /* Database error block               */
short       errsz;       /* Number of characters in message    */

if (err.ERRCODE != SUCCESS)
   ERRMESS (&err, error_msg, &errsz);
```

Sample FORTRAN Program

```
RECORD       /ERRBLOCK/ERR
RECORD       /ERRARRAY/ERROR_MSG
INTEGER*2    ERRSZ
IF (ERROR.ERRCODE .EQ. SUCCESS) THEN GOTO 100
CALL ERRMESS ( ERR, %REF(ERROR_MSG), ERRSZ)
100     CONTINUE
```

FIELDDEFNINFO

Returns database information about a field.

Format

FIELDDEFNINFO(recid, ft, seq, flddefninfo,
    stop, err)

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	recid is the ID of the record

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	Ft is the field tag in the record being examined. The occurrence number portion must be zero or the number of an occurrence in memory. The occurrence number be zero if the field is in the fixed area. The occurrence number may also be zero if any of the following is true: 1) the field does not require a format record, or 2) it is not a ghost field and its format record is not specified by field in the same repeat area, or 3) seq is the history sequence number of the occurrence.

seq

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	History sequence number of occurrence. As described above, seq is not always required.

flddefninfo

Data Type	FIELDDEFN
Access	Output only
Mechanism	Passed by reference
Description	Address of the buffer that will receive the information about the field (see setcim.h)..

stop

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	status code returned by disk history.

err

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	err is the returned InfoPlus.21 error code.

Sample C Program

```
short  stopWord = 0;
long  seqNum = 0;
ERRBLOCK err;
FIELDDEFN fldDefn;


...
FIELDDEFNINFO( recid, ft, seqNum, &fldDefn, &stopWord, &err);
if (err.ERRCODE != SUCCESS)
 printf( "ERROR in FIELDDEFNINFO, code=<%d>\n", err.ERRCODE);
else
{
 printf( "Format record for field tag = %d is %d\n",
  ft, fldDefn.FORMAT_RECORD);
}
```

FIELDINFO

Returns database information about a field.

Format

```
FIELDINFO (recid, ft, datatype, dspchars, inchars, unusacha,
usacha, nowopcha, resizable, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag in the record being examined.

datatype

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*datatype* is the data type indicator as defined in the **setcim.h** include file.

dspchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*dspchars* is the number of characters in the ASCII representation of the field.

inchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*inchars* is the maximum number of characters to allow for data entry.

Unusacha

Data Type:	Byte
Access:	Output only
Mechanism:	Passed by reference
Description:	*unusacha* returns TRUE if the field can be changed when the record is unusable.

usacha

Data Type	Byte
Access	Output only
Mechanism	Passed by reference
Description	*usacha* returns TRUE if the field can be changed when the record is usable.

nowopcha

Data Type	Byte
Access	Output only
Mechanism	Passed by reference
Description	*nowopcha* returns TRUE if the field may be changed by the operator in the current state.

resizable

Data Type	Byte
Access	Output only
Mechanism	Passed by reference
Description	*resizable* returns TRUE if the field is a repeat area sizing field.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID                    */
            ft;         /* Field Tag                    */
short       datatype,           /* Data type == DTYP????        */
            dspchars,           /* ASCII display size       */
            inchars;            /* ASCII input size             */
char        unusacha,   /* Field changeability when unusable    */
            usacha,             /* Field changeability when usable */
            nowopcha,   /* Operator changeability       */
            resizable;          /* Repeat area count field?     */
ERRBLOCK    err;        /* Error return status          */
```

```
FIELDINFO (recid, ft, &datatype, &dspchars, &inchars, &unusacha,
&usacha, &nowopcha, &resizable, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*2   DATATYPE
INTEGER*2   DSPCHARS
INTEGER*2   INCHARS
BYTE        UNUSACHA
BYTE        USACHA
BYTE        NOWOPCHA
BYTE        RESIZABLE
RECORD          /ERRBLOCK/ERR
```

```
CALL FIELDINFO (%VAL(RECID), %VAL(FT), DATATYPE, DSPCHARS,
INCHARS,
UNUSACHA, USACHA, NOWOPCHA, RESIZABLE, ERR)
```

FLDFT

Get field tag of a numbered field in a record and the total number of fields in the record.

Format

```
FLDFT (recid, fldnum, fldftid, numflds)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

fldnum

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*fldnum* is the number of the field in the record.

fldftid

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*fldftid* the field tag.

numflds

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*numflds* is the returned number of fields in the record. In the above example, *numflds* would return 8. If the record ID is invalid, then *numflds* will be 0. If the record ID is valid but *fldnum* exceeds the total number of fields, then *fldftid* returns 0.

Sample C Program

```
long        recid,          /* Recid of record to search    */
            fldnum,                 /* Location in record    */
            fldftid,        /* Returned field tag    */
            numflds;                /* Returned # of fields in record */

FLDFT (recid, fldnum, &fldftid, &numflds);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FLDNUM
INTEGER*4   FLDFTID
INTEGER*4   NUMFLDS

CALL FLDFT (%VAL(RECID), %VAL(FLDNUM), FLDFTID,  NUMFLDS)
```

FOLDERIN

Adds records to a folder if they do not already reside in the folder.

Format

```
FOLDERIN(folderid, records, numrecs, numok, error)
```

Arguments

folderid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*folderid* is the ID of the folder record.

records

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*records* is an array of IDs of records to be added to the folder.

numrecs

Data Type	short
Access	Input only
Mechanism	Passed by value
Description	*numrecs* is the number of records to add to folder.

numok

Data Type	short
Access	Output only
Mechanism	Passed by reference
Description	*numok* is set to *numrecs* if all the specified records were added to the folder or were already in the folder. *numok* is set to zero if *folderid* or *numrecs* is invalid or if the repeat area cannot be extended. *numok* is set to less than *numrecs* if an UNUSABLE record ID is in the *records* array.

error

Data Type	ERRBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#define ARRYSZ 10
long        folderid,          /* Folder record ID          */
            records[ARRYSZ];   /* Array of IDs              */
short       numrecs;           /* Number of records to add*/
short       numok;             /* Number of records added */
ERRBLOCK    err;

FOLDERIN (folderid, records, numrecs, &numok, &err);
```

Sample FORTRAN Program

```
INTEGER*4   FOLDERID
INTEGER*2   ARRYSZ
PARAMETER   10
INTEGER*4   RECORDS(ARRYSZ)
INTEGER*2   NUMRECS
INTEGER*2   NUMOK
RECORD      /ERRBLOCK/ERR

CALL FOLDERIN (%VAL(FOLDERID), RECORDS, %VAL(NUMRECS),NUMOK,
ERR)
```

FOLDEROUT

Removes records from a folder.

Format

```
FOLDEROUT(folderid, records, numrecs, numok, error)
```

Arguments

folderid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*folderid* is the ID of the folder record.

records

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*records* is an array of IDs of records to be removed from the folder.

numrecs

Data Type	short
Access	Input only
Mechanism	Passed by value
Description	*numrecs* is number of records to remove from the folder.

numok

Data Type	short
Access	Output only
Mechanism	Passed by reference
Description	*numok* is set to *numrecs* if all the specified records were removed from the folder or were not in the folder. *numok* is set to zero if *folderid* is invalid.

error

Data Type	ERRBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#define ARRYSZ 10
long      folderid,           /* Folder record ID                */
          records[ARRYSZ]; /* Array of IDs                    */
short     numrecs;            /* Number of records to remove     */
short      numok;             /* Number of records removed       */
ERRBLOCK  err;

FOLDEROUT (folderid, records, numrecs, &numok, &err);
```

Sample FORTRAN Program

```
INTEGER*4   FOLDERID
INTEGER*2   ARRYSZ
PARAMETER   10
INTEGER*4   RECORDS(ARRYSZ)
INTEGER*2   NUMRECS
INTEGER*2   NUMOK
RECORD      /ERRBLOCK/ERR


CALL FOLDEROUT (%VAL(FOLDERID), RECORDS, %VAL(NUMRECS), NUMOK,
ERR)
```

FTNAME2FT

Converts a field name to a field tag. **FTNAM2FT** returns 0 if the field name is invalid.

Format

```
FTNAME2FT(ptbuff, numchars)
```

Returns

long word

FTNAM2FT returns 0 if the field name is invalid, otherwise the field tag is returned.

Arguments

ptbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer containing the field name.

numchar

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchar* is the number of characters in the buffer.

Sample C Program

```
char    *ptbuff = "NAME";    /* Address of the buffer containing the field name
        */
short   numchars = 4;        /* Number of characters in the buffer
        */
long    ft;

ft = FTNAM2FT (ptbuff, numchars);
```

Sample FORTRAN Program

```
CHARACTER   PTBUFF(4)
DATA        PTBUFF/'NAME'/
INTEGER*2   NUMCHARS
DATA        NUMCHARS/4/
INTEGER*4   FT

FT = FTNAM2FT (%REF(PTBUFF), %VAL(NUMCHARS))
```

GETDBPERMIS

Verifies a user's database permissions specified in the *wantedDbPermis* argument. If the user has all of the requested permissions, the *granted* argument is set to TRUE; otherwise it is set to FALSE. The argument *availDbPermis* contains only the permissions that are available to the user.

Format

```
GETDBPERMIS (wantedDbPermis, granted, availDbPermis, err)
```

Arguments

wantedDbPermis

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*wantedDbPermis* contains database permissions to check for.

granted

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*granted* returns TRUE if the caller has all the permissions listed in *wantedDbPermis*; otherwise it returns FALSE

availDbPermis

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*availDbPermis* contains the database permissions available to the user. *availDbPermis is a* subset of *wantedDbPermis*.

err

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#include <setcim.h>
#include <ip21security.h>
#include <stdlib.h>
#include <stdio.h>

main ()
{
ERRBLOCK    err;
ERRARRAY    errmess;
short       errsize;
long        availPermis = AT_PERMIS_NONE;
long        granted = FALSE;

GETDBPERMIS(AT_PERMIS_DB_READ          |
AT_PERMIS_DB_READ_ALL       |
AT_PERMIS_DB_WRITE          |
AT_PERMIS_DB_WRITE_ALL      |
AT_PERMIS_DB_CREATE         |
AT_PERMIS_DB_CREATE_ALL     |
AT_PERMIS_DB_DELETE         |
AT_PERMIS_DB_DELETE_ALL     |
AT_PERMIS_DB_ADMIN          |
AT_PERMIS_DB_CHANGE_SEC     |
AT_PERMIS_DB_CHANGE_SEC_ALL |
AT_PERMIS_DB_CHANGE_DB_SEC,
&granted, &availPermis, &err);
if (err.ERRCODE != SUCCESS)
```

```
{
ERRMESS (&err, errmess, &errsize);
errmess[errsize] = '\0';
printf ("Error: %s\n", errmess);
exit(0);
}

if (granted)
printf(  "Requested access is granted.\n");
else
{
printf(  "Requested access is NOT granted.\n\n");

if ((availPermis & AT_PERMIS_DB_READ) == AT_PERMIS_DB_READ)
printf(  "AT_PERMIS_DB_READ          is     available.\n");
else
printf(  "AT_PERMIS_DB_READ          is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_READ_ALL) ==
AT_PERMIS_DB_READ_ALL)
printf(  "AT_PERMIS_DB_READ_ALL      is     available.\n");
else
printf(  "AT_PERMIS_DB_READ_ALL      is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_WRITE) == AT_PERMIS_DB_WRITE)
printf(  "AT_PERMIS_DB_WRITE         is     available.\n");
else
printf(  "AT_PERMIS_DB_WRITE         is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_WRITE_ALL) ==
AT_PERMIS_DB_WRITE_ALL)
printf(  "AT_PERMIS_DB_WRITE_ALL     is     available.\n");
else
printf(  "AT_PERMIS_DB_WRITE_ALL     is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_CREATE) == AT_PERMIS_DB_CREATE)
```

```
printf(  "AT_PERMIS_DB_CREATE         is     available.\n");
else
printf(  "AT_PERMIS_DB_CREATE         is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_CREATE_ALL) ==
AT_PERMIS_DB_CREATE_ALL)
printf(  "AT_PERMIS_DB_CREATE_ALL     is     available.\n");
else
printf(  "AT_PERMIS_DB_CREATE_ALL     is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_DELETE) == AT_PERMIS_DB_DELETE)
printf(  "AT_PERMIS_DB_DELETE         is     available.\n");
else
printf(  "AT_PERMIS_DB_DELETE         is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_DELETE_ALL) ==
AT_PERMIS_DB_DELETE_ALL)
printf(  "AT_PERMIS_DB_DELETE_ALL     is     available.\n");
else
printf(  "AT_PERMIS_DB_DELETE_ALL     is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_ADMIN) == AT_PERMIS_DB_ADMIN)
printf(  "AT_PERMIS_DB_ADMIN          is     available.\n");
else
printf(  "AT_PERMIS_DB_ADMIN          is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_CHANGE_SEC) ==
AT_PERMIS_DB_CHANGE_SEC)
printf(  "AT_PERMIS_DB_CHANGE_SEC     is     available.\n");
else
printf(  "AT_PERMIS_DB_CHANGE_SEC     is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_CHANGE_SEC_ALL) ==
AT_PERMIS_DB_CHANGE_SEC_ALL)
printf(  "AT_PERMIS_DB_CHANGE_SEC_ALL is     available.\n");
else
```

```
printf(  "AT_PERMIS_DB_CHANGE_SEC_ALL is NOT available.\n");

if ((availPermis & AT_PERMIS_DB_CHANGE_DB_SEC) ==
AT_PERMIS_DB_CHANGE_DB_SEC)
printf(  "AT_PERMIS_DB_CHANGE_DB_SEC  is     available.\n");
else
printf(  "AT_PERMIS_DB_CHANGE_DB_SEC  is NOT available.\n");
}

}  // main
```

GETDBXTIM

Returns the current database time as an Aspen InfoPlus.21 extended timestamp.

Format

```
GETDBXTIM(xtime)
```

Arguments

xtime

Data Type	XTSBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*xtime* is the current Aspen InfoPlus.21 time as an extended timestamp.

Sample C Program

```
XTSBLOCK     xtime;
GETDBXTIM(&xtime);
```

Sample FORTRAN Program

```
RECORD                /XTSBLOCK/XTIME

CALL GETDBXTIM(XTIME)
```

GETFLDPERMIS

This function verifies a user's write permissions against a field in a record. The argument *wantedFldPermis* contains the field write permissions the user wants to check for. If the user has all of the requested field write permissions, the *granted* argument is set to TRUE, otherwise it is set to FALSE. The argument *availFldPermis* contains only the write permissions that the user has on the field.

Format

```
GETFLDPERMIS (recid, ft, wantedFldPermis, granted,
availFldPermis, err)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag to be named in ASCII. The occurrence number is required.

wantedFldPermis

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*wantedFldPermis* contains the field permissions to check for.

granted

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*granted* returns TRUE if the caller has all the permissions listed in *wantedRecPermis*; otherwise *granted* returns false.

availFldPermis

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*availFldPermis* contains the field permissions available to the user. *availFldPermis* is a subset of *wantedFldPermis*.

err

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#include <setcim.h>
#include <ip21security.h>
#include <stdlib.h>
#include <stdio.h>

main ()
{
ERRBLOCK    err;
ERRARRAY    errmess;
short       errsize;
long        availPermis = AT_PERMIS_NONE,
            granted = FALSE,
```

```
                recID,
        ft;
char            *recName = "Test1",
                *ftName = "NAME";

if ( ! INISETC() )
        {
        printf(" SETCIM IS NOT RUNNING !! \n");
        exit(EXIT_FAILURE);
        }

recID = NAME2RECID(  recName, (short)strlen( recName ));
if (recID <= 0)
        {
         printf(" NAME2RECID TEST FAILED 1 \n");
        ENDSETC();
         exit(EXIT_FAILURE);
        }

ft = FTNAME2FT(   ftName, (short)strlen( ftName ));
if (ft == 0)
        {
        printf(  "ERROR in FTNAME2FT\n" );
        ENDSETC();
         exit(EXIT_SUCCESS);
         }

GETFLDPERMIS(  recID, ft, AT_PERMIS_REC_ALL, &granted,
        &availPermis, &err );
if (err.ERRCODE != SUCCESS)
        {
        RRMESS (&err, errmess, &errsize);
        errmess[errsize] = '\0';
        rintf ("Error: %s\n", errmess);
        NDSETC();
         exit(0);
```

```
        }

if (granted)
        printf(  "Requested access is granted.\n");
else
        {
        printf(  "Requested access is NOT granted.\n\n");

        if ((availPermis & AT_PERMIS_REC_READ) ==
AT_PERMIS_REC_READ)
                printf(  "AT_PERMIS_REC_READ is available.\n");
        else
                printf(  "AT_PERMIS_REC_READ is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_WRITE_GENERAL) ==
AT_PERMIS_REC_WRITE_GENERAL)
printf(  "AT_PERMIS_REC_WRITE_GENERAL  is     available.\n");
else
printf(  "AT_PERMIS_REC_WRITE_GENERAL  is NOT
available.\n");

if ((availPermis & AT_PERMIS_REC_WRITE_RESTRICTED) ==
AT_PERMIS_REC_WRITE_RESTRICTED)
printf(  "AT_PERMIS_REC_WRITE_RESTRICTED is    available.\n");
else
printf(  "AT_PERMIS_REC_WRITE_RESTRICTED is NOT
available.\n");

if ((availPermis & AT_PERMIS_REC_WRITE_SYSTEM) ==
AT_PERMIS_REC_WRITE_SYSTEM)
printf(  "AT_PERMIS_REC_WRITE_SYSTEM   is     available.\n");
else
printf(  "AT_PERMIS_REC_WRITE_SYSTEM   is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_DELETE) ==
AT_PERMIS_REC_DELETE)
```

```
printf(  "AT_PERMIS_REC_DELETE         is     available.\n");
else
printf(  "AT_PERMIS_REC_DELETE         is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_CHANGE_SEC) ==
AT_PERMIS_REC_CHANGE_SEC)
printf(  "AT_PERMIS_REC_CHANGE_SEC     is     available.\n");
else
printf(  "AT_PERMIS_REC_CHANGE_SEC     is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_CREATE) ==
AT_PERMIS_REC_CREATE)
printf(  "AT_PERMIS_REC_CREATE         is     available.\n");
else
printf(  "AT_PERMIS_REC_CREATE         is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_ACT) == AT_PERMIS_REC_ACT)
printf(  "AT_PERMIS_REC_ACT            is     available.\n");
else
printf(  "AT_PERMIS_REC_ACT            is NOT available.\n");
}

ENDSETC();

}  // main
```

GETFTDB

Returns the field name given the record ID and field tag.

Format

```
GETFTDB(recid, ft, ftbuff, numchars)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag to be named in ASCII. The occurrence number is required.

ftbuff

Data Type	FTNMARR
Access	Output only
Mechanism	Passed by reference
Description	*ftbuff* is the character buffer containing the returned field name, including the occurrence if *ftbuff* is in a repeat area.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the size of the field name.

Sample C Program

```
long        recid,      /* Record ID              */
            ft;         /* Field Tag              */

FTNMARR     ftbuff;     /* Output buffer for name    */
short       numchars;   /*  Size of buffer used */

GETFTDB (recid, ft, ptbuff, &numchars);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
RECORD              /FTNMARR/PTBUFF
INTEGER*2    NUMCHARS

CALL GETFTDB (%VAL(RECID), %VAL(FT), %REF(PTBUFF),  NUMCHARS)
```

GETNAMDB

Returns the record name given the record ID.

Format

```
GETNAMDB(recid, nambuff, numchars)
```

Arguments

recid

Data Type	long word
Access	Input/Output
Mechanism	Passed by reference
Description	*recid* is the ID of the record. If invalid, *recid* is set to 0.

nambuff

Data Type	NAMEARR
Access	Output only
Mechanism	Passed by reference
Description	*nambuff* is the character buffer containing the returned record name.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the size of the record name.

Sample C Program

```
long        recid;      /* Record ID          */
NAMEARR     nambuff;            /* Output buffer for name   */
short       numchars;   /* Size of buffer used        */

GETNAMDB (&recid, nambuff, &numchars);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
RECORD      /NAMARR/NAMBUFF
INTEGER*2   NUMCHARS

CALL GETNAMDB(RECID, %REF(NAMBUFF), NUMCHARS)
```

GETNMFDB

Returns the record and field names given a record ID and field tag.

Format

```
GETNMFDB(recid, ft, nmftbuff, numchars)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag. The occurrence number is required.

nmftbuff

Data Type	NAMFTARR
Access	Output only
Mechanism	Passed by reference
Description	*nmftbuff* is the character buffer containing the returned record and field names, including the occurrence if in a repeat area.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the size of the buffer used.

Sample C Program

```
long        recid,      /* Record ID          */
            ft;         /* Field Tag          */
NAMFTARR    nmftbuff;           /* Output buffer for names  */
short       numchars;   /* Size of buffer used        */

GETNMFDB (recid, ft, nmftbuff, &numchars);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
RECORD      /NAMFTARR/NMFTBUFF
INTEGER*2   NUMCHARS

CALL GETNMFDB ( %VAL(RECID), %VAL(FT), %REF(NMFTBUFF), NUMCHARS)
```

GETRECLIST

Returns a list of record IDs and names that meet specific criteria.

Format

```
GETRECLIST(lastrecid, rectype, alphaorder, grouplist, groupsize,
viewreq, modifyreq, maxrecs, recids, recusabs, recnames,
namesizes, numrecs)
```

Arguments

lastrec

Data Type	long word
Access	Input/Output
Mechanism	Passed by reference
Description	*lastrec* the ID of the last record returned by a previous call to **GETRECLIST**. To build a new list, *lastrec* should be set to zero. If *lastrec* equals 0 on exit, the list is complete. If *lastrec* equals -1 there is an error in the parameters.

rectype

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*rectype* is the record ID of a definition record or a record type identifier from the table below:

Record Type	Identifier
Any Record	RTYPANYRECORD
External Task Record	RTYPEXTASK
Field Name Record	RTYPFLDNAME
Definition Record	RTYPDEFINE
Select Descriptor Record	RTYPSELECT
Disk History Record	RTYPDSKHIST
History Summary Line Record	RTYPHSUMLIN
Pseudo Summary Line Record	RTYPPSUMLIN
Normal Summary Line Record	RTYPNSUMLIN
Integer Format Record	RTYPIFORMAT
Real Format Record	RTYPRFORMAT
Timestamp Format Record	RTYPTFORMAT
Detail Display Record	RTYPDETDSPLY

Record Type	Identifier
External Task Record Definition Record	RTYPDEFEXTASK
Field Name Record Definition Record	RTYPDEFFLDNAME
Definition Record Definition Record	RTYPDEFDEFINE
Select Descriptor Record Definition Record	RTYPDEFSELECT
Disk History Record Definition Record	RTYPDEFDSKHIST

alphaorder

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	If *alphaorder* is zero, the records will be returned in record ID order. Otherwise, the records will be returned in alphabetical order.

grouplist

Data Type	byte array
Access	Input only
Mechanism	Passed by reference
Description	*grouplist* is an array of bytes identifying a user's group membership.

groupsize

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*groupsize* is the length of *grouplist* in bytes.

viewreq

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*viewreq* is nonzero if the users group must have view access to the record or zero if view access is not required.

modifyreq

Data Type	long word

Access	Input only
Mechanism	Passed by value
Description	*modifyreq* is nonzero if the users group must have modify access to the record or zero if modify access is not required.

maxrecs

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*maxrecs* is the maximum number of records to include in the list.

recids

Data Type	long word array
Access	Output only
Mechanism	Passed by reference
Description	*recids* is an array of record IDs returned by `GETRECLIST`. *Recids* must have at least *maxrecs* elements.

recusabs

Data Type	byte array
Access	Output only
Mechanism	Passed by reference
Description	*recusabs* is an array indicating the usability of the records in *recids*. For each record, the corresponding element of *recusabs* is zero if the record is unusable and nonzero if the record is usable. *recusabs* must have at least *maxrecs* elements.

recnames

Data Type	NAMEARR array
Access	Output only
Mechanism	Passed by reference
Description	*recnames* is an array containing the record names of the records in *recids*. *recnames* must have at least *maxrecs* elements.

namesizes

Data Type	short word array
Access	Output only
Mechanism	Passed by reference
Description	*namesizes* is an array containing the lengths of the record names in *recnames*. *namesizes* must have at least *maxrecs* elements.

numrecs

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*numrecs* is the actual number of records returned in *recids*. If *numrecs* = *maxrecs* after calling **GETRECLIST**, call **GETRECLIST** again with *lastrec* containing the last record ID in *recids*.

Sample C Program

```
long            lastrec,                /*  The last record ID returned  from
        */
                                /*  GETRECLIST                          */
                rectype,                /*  Type of record desired          */
unsigned char   grouplist[8];   /*  Group membership array              */
long            recids[100];    /*  Array to be filled with record IDs  */
char            recusabs[100];          /*  Array indicating each record's  */
                                /* usability                           */
NAMEARR         recnames[100];          /*  Array to be filled with record names
        */
short           namesizes[100];         /*  Array of sizes of record names  */
long            numrecs;                /*  Number of records received      */


rectype = RTYPDEFINE;
lastrec = 0L;
do
{    GETRECLIST(&lastrec, rectype, 1, grouplist, 8, 0, 0, 100, recids,
                    recusabs,recnames, namesizes, &numrecs );
} while (lastrec> 0L);          /**** Time: between lastrec & 0 ****/
```

Sample FORTRAN Program

```
INTEGER*4           LASTREC
INTEGER*4           RECTYPE
BYTE                GROUPLIST(8)
```

```
INTEGER*4          RECIDS(100)
BYTE               RECUSABS(100)
RECORD                   /NAMEARR/RECNAMES(100)
INTEGER*2          NAMSIZES(100)
INTEGER*4          NUMRECS

LASTREC = 1
RECTYPE = RTYPDEFINE

DO WHILE (LASTREC .GT. 0)
    CALL GETRECLIST(LASTREC, %VAL(RECTYPE), %VAL(1), (8),%VAL(0),
        %VAL(0),%VAL(100),RECIDS,RECUSABS,RECNAMES,
        NAMSIZES,NUMRECS)
END DO
```

GETRECPERMIS

Verifies a user's record permissions specified in the *wantedRecPermis* argument. If the user has all of the requested permissions, the *granted* argument is set to TRUE; otherwise, it is set to FALSE. The argument *availRecPermis* contains only the permissions that the user has on the record.

Format

```
GETRECPERMIS (recid, wantedRecPermis, granted, availRecPermis,
err)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is ID of the record containing the field.

wantedRecPermis

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*wantedRecPermis* contains the record permissions to check for.

granted

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*granted* returns TRUE if the caller has all the permissions listed in *wantedRecPermis*; otherwise *granted* returns false.

availRecPermis

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*availRecPermis* contains the record permissions available to the user. *availRecPermis* is a subset of *wantedRecPermis*.

err

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#include <setcim.h>
#include <ip21security.h>
#include <stdlib.h>
#include <stdio.h>

main ()
{
ERRBLOCK    err;
ERRARRAY    errmess;
short               errsize;
```

```
long                    availPermis = AT_PERMIS_NONE,
                granted = FALSE,
recID,
i;
char    recName[10] = "TEST1";

if ( ! INISETC() )
{
printf(" SETCIM IS NOT RUNNING !! \n");
exit(EXIT_FAILURE);
}

recID = NAME2RECID(  recName, (short)strlen( recName ));
if (recID <= 0)
{
printf(" NAME2RECID TEST FAILED 1 \n");
ENDSETC();
exit(EXIT_FAILURE);
}

for (i=0; i<3; i++)
{
availPermis = AT_PERMIS_NONE,
granted = FALSE,

GETRECPERMIS(  recID,
AT_PERMIS_REC_READ          |
AT_PERMIS_REC_WRITE         |
AT_PERMIS_REC_DELETE        |
AT_PERMIS_REC_CHANGE_SEC    |
AT_PERMIS_REC_CREATE        |
AT_PERMIS_REC_ACT,
&granted,
&availPermis,
&err );
if (err.ERRCODE != SUCCESS)
```

```
{
ERRMESS (&err, errmess, &errsize);
errmess[errsize] = '\0';
printf ("Error: %s\n", errmess);
ENDSETC();
exit(0);
}

if (granted)
printf(  "Requested access is granted.\n");
else
{
printf(  "Requested access is NOT granted.\n\n");

if ((availPermis & AT_PERMIS_REC_READ) ==
AT_PERMIS_REC_READ)
printf(  "AT_PERMIS_REC_READ is available.\n");
else
printf(  "AT_PERMIS_REC_READ is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_WRITE_GENERAL) ==
        AT_PERMIS_REC_WRITE_GENERAL)
printf(  "AT_PERMIS_REC_WRITE_GENERAL  is available.\n");
else
printf(  "AT_PERMIS_REC_WRITE_GENERAL  is NOT
available.\n");

if ((availPermis & AT_PERMIS_REC_WRITE_RESTRICTED) ==
AT_PERMIS_REC_WRITE_RESTRICTED)
printf(  "AT_PERMIS_REC_WRITE_RESTRICTED is    available.\n");
else
printf(  "AT_PERMIS_REC_WRITE_RESTRICTED is NOT
available.\n");

if ((availPermis & AT_PERMIS_REC_WRITE_SYSTEM) ==
AT_PERMIS_REC_WRITE_SYSTEM)
```

```
printf(  "AT_PERMIS_REC_WRITE_SYSTEM   is     available.\n");
else
printf(  "AT_PERMIS_REC_WRITE_SYSTEM   is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_DELETE) ==
AT_PERMIS_REC_DELETE)
printf(  "AT_PERMIS_REC_DELETE         is     available.\n");
else
printf(  "AT_PERMIS_REC_DELETE         is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_CHANGE_SEC) ==
AT_PERMIS_REC_CHANGE_SEC)
printf(  "AT_PERMIS_REC_CHANGE_SEC     is     available.\n");
else
printf(  "AT_PERMIS_REC_CHANGE_SEC     is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_CREATE) ==
AT_PERMIS_REC_CREATE)
printf(  "AT_PERMIS_REC_CREATE         is     available.\n");
else
printf(  "AT_PERMIS_REC_CREATE         is NOT available.\n");

if ((availPermis & AT_PERMIS_REC_ACT) == AT_PERMIS_REC_ACT)
printf(  "AT_PERMIS_REC_ACT            is     available.\n");
else
printf(  "AT_PERMIS_REC_ACT            is NOT available.\n");
}
}

ENDSETC();

}  // main
```

GETWRITELEVEL

Gets the write-level permission value of the specified field in the record. Write level is used in coordination with the record *write* permissions to restrict a user's ability to modify a field. To read a field's write level, the user must have a read permission to the record.

Format

```
GETWRITELEVEL (recid, ft, intdata, writeLevel, err)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the tag of the field to find the write level from.

write level

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*write level* is the write level of the field. Possible values are: • AT_PERMIS_NONE (no permission) • AT_PERMIS_REC_WRITE_GENERAL • AT_PERMIS_REC_WRITE_RESTRICTED • AT_PERMIS_REC_WRITE_SYSTEM

err

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#include <setcim.h>
#include <ip21security.h>
#include <stdlib.h>
#include <stdio.h>

main ()
{
ERRBLOCK    err;
ERRARRAY    errmess;
short       errsize;
long        writeLevel = WRITE_NONE,
recID,
ft;
char        *recName = "Test1",
*ftName = "NAME";
if ( ! INISETC() )
{
printf(" SETCIM IS NOT RUNNING !! \n");
exit(EXIT_FAILURE);
}
recID = NAME2RECID(  recName, (short)strlen( recName ));
if (recID <= 0)
{
printf(" NAME2RECID TEST FAILED 1 \n");
ENDSETC();
exit(EXIT_FAILURE);
```

```
}
ft = FTNAME2FT(   ftName, (short)strlen( ftName ));
if (ft == 0)
{
printf(  "ERROR in FTNAME2FT\n" );
ENDSETC();
exit(EXIT_SUCCESS);
}
GETWRITELEVEL( recID, ft, &writeLevel, &err );
if (err.ERRCODE != SUCCESS)
{
ERRMESS (&err, errmess, &errsize);
errmess[errsize] = '\0';
printf ("Error: %s\n", errmess);
ENDSETC();
exit(0);
}
if ((writeLevel & WRITE_NONE) == WRITE_NONE)
printf(  "Write level =  WRITE_NONE.\n");
if ((writeLevel & WRITE_GENERAL) == WRITE_GENERAL)
printf(  "Write level =  WRITE_GENERAL.\n");
if ((writeLevel & WRITE_RESTRICTED) == WRITE_RESTRICTED)
printf(  "Write level =  WRITE_RESTRICTED.\n");
if ((writeLevel & WRITE_SYSTEM) == WRITE_SYSTEM)
printf(  "Write level =  WRITE_SYSTEM.\n");


ENDSETC();


}  // main
```

HISOLDESTOK

Obtains and/or changes the oldest allowed time for the history repeat area of a record.

Format

```
HISOLDESTOK (id, ft, newoldest, oldoldest, err)
```

Arguments

id

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*id* is ID of the record.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of a field in the repeat area or the repeat area count field tag.

newoldest

Data Type	integer
Access	Input only
Mechanism	Passed by value
Description	*newoldest* is the new oldest time allowed in seconds since 1970. To obtain the current oldest time allowed, *newoldest* can be called with *newoldest* = -1.

oldoldest

Data Type	integer
Access	Output only
Mechanism	Passed by reference
Description	*oldoldest* is the returned previous oldest time allowed.

err

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
#include <setcim.h>

void SET_OLDEST_TIME_TO_48_HOURS_AGO(void)
{
        long id = NAME2RECID("TestRecord",10);
        long ft = FTNAME2FT("1 IP_TREND_TIME", 15);
        int newoldest;
        int oldoldest;
        ERRBLOCK err;
        XTSBLOCK current_xts;
        XUSTS current_time;

        GETDBXTIM(&current_xts);
        XTS2XUST(&current_xts, &current_time);
        newoldest = current_time.secs - 48*60*60;
        HISOLDESTOK(id, ft, newoldest, &oldoldest, &err);
        if(err.ERRCODE)
          printf("Error calling HISOLDESTOK\n");
}
```

I2ASCIIDB

I2ASCIIDB converts an integer to ASCII in the format of a database field.

Format

```
I2ASCIIDB(recid, ft, intdata, ptbuff, maxchars, numchars, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID for the integer to be converted.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of an integer field. This field's format will be used to format the integer value in *intdata*.

intdata

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*intdata* is the long word integer value to be converted to ASCII.

ptbuff

Data Type	character array
Access	input only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer to receive the ASCII data.

maxchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*maxchars* specifies the maximum number of characters in the buffer.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the number of characters which are normally written to a buffer. If *numchars* is greater than *maxchars*, only *maxchars* characters are used.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /*  Record ID                    */
            ft,         /*  Field tag                    */
            indata;     /*  long word integer value      */
char        ptbuff[20]; /*  Address of the buffer to receive the   */
                        /*  ASCII data                   */
short       maxchars,   /*  Max number of chars in the buffer      */
            numchars;   /*  Number of chars which are    */
                        /*  normally written to a buffer           */
ERRBLOCK    err;

I2ASCIIDB (recid, ft, indata, ptbuff, maxchars, &numchars, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
INTEGER*4    INDATA
CHARACTER*20       PTBUFF
INTEGER*2    MAXCHARS
INTEGER*2    NUMCHARS
RECORD              /ERRBLOCK/ERR

CALL I2ASCIIDB ( %VAL(RECID),%VAL(FT),%VAL(INDATA),%REF(PTBUFF),
                %VAL(MAXCHARS),NUMCHARS,ERR)
```

IDFT2DB

Writes a record ID and field tag to a database field.

Format

```
IDFT2DB(recid, ft, idftdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record containing the field to write into.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field to write into.

idftdata

Data Type	IDANDFT
Access	Input only
Mechanism	Passed by reference
Description	*idftdata* is the buffer containing the record ID and field tag values.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long            recid,  /*  Record ID                  */
                ft;     /*  Field Tag                  */
IDANDFT                 idftdata;       /*  Output buffer for ID and tag     */
ERRBLOCK    err;
IDFT2DB (recid, ft, &idftdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
RECORD              /IDANFT/IDFTDATA
RECORD              /ERRBLOCK/ERR

CALL IDFT2DB ( %VAL(RECID), %VAL(FT), IDFTDATA, ERR)
```

INISETC

Starts Aspen InfoPlus.21 support in a program. This routine must be called before any Aspen InfoPlus.21 access routines are called.

Returns

word

Returns a one (1) if Aspen InfoPlus.21 is up, and a zero (0) if Aspen InfoPlus.21 is down.

Format

```
INISETC()
```

Arguments

None

Sample C Program

```
main()
{
   if ( !INISETC() )
   {   /*  InfoPlus.21 is DOWN or no READ access*/
       exit():
   }
  ENDSETC();
}
```

Sample FORTRAN Program

```
PROGRAM MYPROG

      IF( INISETC() .EQ. 0 ) THEN
C     InfoPlus.21 is down
      CALL  EXIT
      ENDIF

      CALL  ENDSETC()
      CALL  EXIT
      END
```

INITCONS

Initializes an interface for interprocess communications between an active GCS/InfoPlus.21 console server process and the calling application program. This routine should always be paired with a subsequent call to the routine **ENDCONS**.

Format

```
INITCONS (console, descriptor, error)
```

Arguments

console

Data Type	long word
Access	input only
Mechanism	Passed by value
Description	*console* is the record ID of the console task record in use by the GCS/InfoPlus.21 console session of interest.

descriptor

Data Type	Console
Access	output only
Mechanism	Passed by reference
Description	*descriptor* is a reference to a structure that identifies the interface that has been initialized. This structure is needed to call other GCS/InfoPlus.21 console access routines. The data type Console is defined in the header file console.h.

error

Data Type	ERRBLOCK
Access	output only
Mechanism	Passed by reference
Description	*error* indicates whether or not the initialization was successful. If *error*.ERRCODE is not set to SUCCESS then the structure referenced by descriptor cannot be used. The following values of *error*.ERRCODE are possible: NOREC INVEXTSK OPSCGCSI RDSCGCSI RFSCGCSI WRSCGCSI

Sample C Program

```
#include <setcim.h>
#include <console.h>

long            console = CONSOLE_TASK_RECID;
Console         descriptor;
ERRARRAY  error_msg;
ERRBLOCK  error;
short           errsz;

/*   Verify that InfoPlus.21 is running  */

if (INISETC ())
{
    INITCONS (console, &descriptor, &error);
    if (error.ERRCODE == SUCCESS)
    {
     <can call other GCS/InfoPlus.21 console access routines here>

        /*  Terminate interface with console session */
        ENDCONS (&descriptor, &error);

     }

    /*  Handle error returned from either INITCONS or ENDCONS  */

    else
    {   ERRMESS (&error, error_msg, &errsz);
        error_msg[errsz] = '\0';
        printf ("%s\n", error_msg);
    }
```

```
    /*  Terminate interface with InfoPlus.21 */
    ENDSETC();
}
else
    printf ("InfoPlus.21 is not up!\n");
```

Sample FORTRAN Program

```
      INCLUDE  setcim.inc
      INCLUDE  console.inc

      INTEGER*4 console;
      RECORD    Console/  descriptor;
      RECORD     /ERRARRAY/ error_msg;
      RECORD     /ERRBLOCK/ error;
      INTEGER*2  errsz;

      DATA console /892/

C  Verify that InfoPlus.21 is up
 IF (INISETC () .NE. 0) THEN
       CALL INITCONS (%VAL(console), descriptor, error)
       IF (error.ERRCODE .EQ. SUCCESS) THEN
          <can call other GCS/InfoPlus.21 console access routines here>
 C       Terminate the interface with the console session
          CALL ENDCONS (descriptor, error)
       ENDIF
C
C  Handle error returned from either INITCONS or ENDCONDS
C
       IF (error.ERRCODE .NE. SUCCESS) THEN
          CALL ERRMESS (error, %REF(error_msg), errsz)
          PRINT 5, error_msg(1:errsz)
       ENDIF
C
C  Terminate interface with InfoPlus.21
C
       CALL ENDSETC ()
     ELSE
       PRINT *, 'InfoPlus.21 is not up'
     ENDIF

5     FORMAT (' ', A)
```

INSOCCS

Inserts new occurrences between two existing occurrences changing the occurrence numbers of all occurrences from the oldest to the point of insertion. Data may be specified to be written into fields in the new occurrences. Any fields in the new occurrences that do not have data specified are initialized to their default values.

The point of insertion is specified by the occurrence number. In other words, occurrences that are before the one specified by the occurrence number are

not changed; all occurrences after the insertion point, including the one specified by the occurrence number, are shifted down.

Format

```
INSOCCS(recid, ft, numfts, fts, datats, numoccs, occnum, ptdatas,
occsinserted, occsok, ftsok, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the Record ID of the record being accessed.

ft

Data Type	long word
Access	Input Only
Mechanism	Passed by Value
Description	*ft* is a Field tag of a field in the record's nonhistorical repeat area (records can have more than one repeat area). To be a valid field the occurrence number needs to be specified, normally set to one.

numfts

Data Type	short word
Access	Input only
Mechanism	Passed by Value
Description	*numfts* is the number of fields per new occurrence that are to be updated with new values. If *numfts* is zero, then all the fields in the new occurrences will have default values.

fts

Data Type	long word
Access	Input Only
Mechanism	Passed by reference
Description	*fts* is an array containing the tags of all fields in the occurrences to update.

datats

Data Type	short word
Access	Input only
Mechanism	Passed by reference
Description	*datats* is an array containing the respective datatypes of all the fields in *fts*.

numoccs

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*numoccs* is the number of occurrences to be inserted by this routine.

occnum

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*occnum* is the occurrence numbers to insert the occurrences.

ptdatas

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*ptdatas* is an array of pointers to data arrays. There is a data array for each field requested. Each successive element in a data array relates to the respective occurrence.

Occsinserted

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*occsinserted* is the number of occurrences inserted successfully.

occsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	Normally *occsok* will be equal *occsinserted* if *numfts* is nonzero, zero otherwise. If there is an error and if *ftsok* is less than *numfts*, *occsinserted* occurrences will have been written with the first *ftsok* fields and *occsok* occurrences will have been written with the next field.

ftsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*ftsok* is the number of fields returned and equals *numfts*, unless there has been an error.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned InfoPlus.21 error code.

Sample C Program

```
#define NUMTAGS 3  /*  Number of fields to update per */
                   /*  inserted occurrence             */
#define ARRSZ 5          /*  Usually equals the #occur to insert     */
float   realarry[ARRSZ];
long    intarry[ARRSZ],
        timearry[ARRSZ];
```

```
long    recid,          /*  Record ID                          */
        ft,             /*  Field in non–history repeat area   */
        fts[NUMTAGS],       /*  List of the field tags to update per occurrence
        */
        numoccs = ARRSZ, /*  Number of occurrences to be inserted    */
long    occnum,             /*  Insertion point                    */
short   occsok,             /*  Returned number of occurrences updated ok
*/
        occsinserted,  /*  Returned number of occurrences inserted    */
        numfts,             /*  Number of field tags in the list   */
        ftsok,          /*  Returned number of valid field tags    */

datats[NUMTAGS] = { DTYPREAL, DTYPLONG, DTYPTIME },
ptdatas[NUMTAGS] = {        /*  Array of pointers to data
*/
        (short *) realarry,     /*  One array for each field tag to be updated
*/
        (short *) intarry,
        (short *) timearry };

ERRBLOCK    err;            /*  InfoPlus.21 error return structure     */

INSOCCS (recid, ft, numfts, fts, datats, numoccs, occnum, ptdatas,
        &occsinserted, &occsok, &ftsok, &err);
```

Sample FORTRAN Program

```
INTEGER*2   NUMTAGS
INTEGER*2   ARRSZ
PARAMETER   ( NUMTAGS = 3 )
PARAMETER   ( ARRYSZ = 5 )
REAL*4              REALARRY(ARRSZ)
INTEGER*4   INTARRY(ARRSZ)
INTEGER*4   TIMEARRY(ARRSZ)
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*4   FTS(NUMTAGS)
INTEGER*4   NUMOCCS
INTEGER*4   OCCNUM
INTEGER*2   OCCSOK
INTEGER*2   OCCSINSERTED
INTEGER*2   NUMFTS
INTEGER*2   FTSOK
INTEGER*2   DATATS(NUMTAGS)
INTEGER*4   PTDATAS(NUMTAGS)
RECORD              /ERRBLOCK/ERR

NUMOCCS     = ARRSZ
NUMFTS              = NUMTAGS
DATATS(1)   = DTYPREAL
DATATS(2)   = DTYPLONG
```

```
DATATS(3)    = DTYPTIME
PTDATAS(1)   = %LOC(REALARRY(1))
PTDATAS(2)   = %LOC(INTARRY(1))
PTDATAS(3)   = %LOC(TIMEARRY(1))

CALL INSOCCS ( %VAL(RECID), %VAL(FT), %VAL(NUMFTS),
               %REF(FTS), %REF(DATATS), %VAL(NUMOCCS),
               %VAL(OCCNUM), %REF(PTDATAS),
               OCCSINSERTED, OCCSOK, FTSOK, ERR)
```

LOGMESS

Adds a message to a log record.

Format

```
LOGMESS(recid, logctrlid, ptmess, numchars, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is zero if messages are to be logged in all log records listed in the log control record specified by logctrlid. If *recid* is nonzero, it is the ID of a record containing fields to be tested to determine which log records to update. As an adjunct to the normal Aspen InfoPlus.21 logging function, this functionality allows the caller to indicate the record and field values to which the message applies. If the record indicated by *recid* contains a condition field listed in the log control record, the value of this field is compared against unallowed values specified in the condition record associated with the field. If the record indicated by *recid* does not contain a condition field, or if the unallowed conditions do not occur, no message is written to the log record. For more information about log control and condition records, see the **Application Basics** chapter in the *Aspen InfoPlus.21 User's Manual*.

logctrlid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*logctrlid* is the record ID of the log control record (defined against LogControlDef).

ptmess

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptmess* is an array that holds the message.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* is the number of characters in the message.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,          /*  ID of a record to be tested              */
            logctrlid,              /* Record ID of a log control record*/
short       numchars;     /* Number of character in the message     */
char        ptmess[112]; /* Buffer holding the message to add      */
ERRBLOCK    err;

LOGMESS (recid, logctrlid, ptmess, numchars, &err)
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    LOGCTRLID
INTEGER*2    NUMCHARS
CHARACTER*112      PTMESS
RECORD             /ERRBLOCK/ERR

CALL LOGMESS( %VAL(RECID), %VAL(LOGCTRLID), %REF(PTMESS),
              %VAL(NUMCHARS), ERR)
```

LONG2DB

Writes a long integer to the database (data type = DTYPLONG).

Format

```
LONG2DB(recid, ft, intdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field to write into.

intdata

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*indata* is the buffer containing the integer value.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID              */
            ft,         /* Field Tag              */
            intdata;            /* Integer data to write to database*/
ERRBLOCK    err;
LONG2DB (recid, ft, intdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*4   INTDATA
RECORD              /ERRBLOCK/ERR

CALL LONG2DB( %VAL(RECID), %VAL(FT), %VAL(INTDATA), ERR)
```

MAKUNUSA

Makes a record unusable. The record cannot be referenced by other records.

Format

```
MAKUNUSA(recid, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,  /* ID of the record to make unusable */
ERRBLOCK    err;

MAKUNUSA (recid, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
RECORD              /ERRBLOCK/ERR

CALL MAKUNUSA ( %VAL(RECID), ERR)
```

MAKUSABL

Makes record usable.

Format

```
MAKUSABL(recid, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,  /* ID of the record to make usable */
ERRBLOCK    err;

MAKUSABL (recid, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
RECORD              /ERRBLOCK/ERR

CALL MAKUSABL ( %VAL(RECID), ERR)
```

MRDBOCCS

Reads multiple occurrences of multiple fields from a repeat area in one database record.

Format

```
MRDBOCCS( recid, numfts, ptfts, ptdtypes, frstoc, lastoc,
ptdatas, occsok, ftsok, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the repeat area.

numfts

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numfts* is the number of fields to read from each occurrence.

ptfts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*ptfts* is an array containing the field tags of the fields to read from each occurrence. The occurrence number of each FT should be zero and there should be *numfts* elements in the array.

ptdtypes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*ptdtypes* is an array that identifies the data type of each field to read. Each element in the array is a data type as defined in the **setcim.h** include file. For example, if the fifth field is a 16-bit signed integer, then the fifth element of *ptdtypes* is DTYPSHRT.

frstoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*frstoc* is the starting occurrence number to be read.

l**astoc**

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*lastoc* is the ending occurrence number to be read.

ptdatas

Data Type	array of addresses
Access	Output only
Mechanism	Passed by reference
Description	*ptdatas* is an array of addresses. Each address is the location of an array in which to store the values read from the occurrences of a specific field. The array must be large enough to contain all of the occurrences read from a field of the type specified in *ptdtypes*.

occsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*occsok* is the actual number of occurrences read.

ftsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*ftsok* is the actual number of fields read. Unless there is an error, *ftsok* will equal *numfts*.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code. The error code is as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /*  ID of the record                    */
            ptfts[3];           /*  Field Tags in the repeat area   */
short       frstoc,     /*  Starting occurrence to read         */
            lastoc,     /*  Ending occurrence to read           */
            numfts,             /*  Number of fields to read        */
```

```
                occsok,                 /*  Number of occurrences read     */
                ftsok;          /*  Number of fields read           */
short           ptdtypes[3];    /*  Data types for each value to read      */
short           field1[200];    /*  Destination for short data             */
long            field2[200];    /*  Destination for long data       */
float           field3[200];    /*  Destination for float data      */
short           *ptdatas[3];    /*  Destination of data read        */
ERRBLOCK        err;            /*  Error for record                */
ptdtypes[0] = DTYPSHRT;
ptdtypes[1] = DTYPLONG;
ptdtypes[2] = DTYPREAL;
ptdatas[0]   = field1;
ptdatas[1]   = field2;
ptdatas[2]   = field3;

MRDBOCCS (recid, numfts, ptfts, ptdtypes, frstoc, lastoc, ptdatas, &occsok,
&ftsok,&err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   PTFTS(3)
INTEGER*2   FRSTOC
INTEGER*2   LASTOC
INTEGER*2   NUMFTS
INTEGER*2   PTDTYPES(3)
INTEGER*4   PTDATAS(3)
INTEGER*2   OCCSOK
INTEGER*2   FTSOK
INTEGER*2   FIELD1(200)
INTEGER*4   FIELD2(200)
REAL*4              FIELD3(200)
RECORD              /ERRBLOCK/ERR
PTDTYPES(1) = DTYPSHRT
PTDTYPES(2) = DTYPLONG
PTDTYPES(3) = DTYPREAL
PTDATAS(1) = %LOC(FIELD1)
PTDATAS(2) = %LOC(FIELD2)
PTDATAS(3) = %LOC(FIELD3)

CALL MRDBOCCS(%VAL(RECID),%VAL(NUMFTS),%REF(PTFTS),
        %REF(PTDTYPES),%VAL(FRSTOC), %VAL(LASTOC),
%REF(PTDATAS),OCCSOK, FTSOK, ERR)
                .
```

MRDBVALS

Reads multiple fields from multiple records in the database.

Format

```
MRDBVALS(numvalus, fields, ptdtypes, ptdatas, numok, error)
```

Arguments

numvalus

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numvalus* is the number of values to be read.

fields

Data Type	IDANDFT array
Access	Input only
Mechanism	Passed by reference
Description	*fields* is an array of record and field IDs to be read.

ptdtypes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*ptdtypes* is the address of an array of the respective data types of the field in *fields* (i.e., *ptdtypes*[1] is the data type of field *fields*[1]).

ptdatas

Data Type	type aligned address
Access	Output only
Mechanism	Passed by reference
Description	*ptdatas* is the address of the destination of the data. Values are consecutive rounded to word boundaries. As the list of fields often refers to different types and sizes of data, *ptdatas* is usually a structure.

numok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numok* is the actual number of values read.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
IDANDFT    fields[3];        /*  List of record & field IDs to be read     */
short      numvalus,/*  Number of fields to read          */
           ptdtypes[3],      /*  List of field tags' data types     */
           numok;    /*  Number of values successfully read     */
ERRBLOCK   err;              /*  Error return status               */

struct  {    short field1;     /*  Destination of data */
             long  field2;
             float field3;
        }  ptdatas;


ptdtypes[0] = DTYPSHRT;
ptdtypes[1] = DTYPLONG;
ptdtypes[2] = DTYPREAL;

MRDBVALS(numvalus, fields, ptdtypes, &ptdatas, &numok, &err);
```

Sample FORTRAN Program

```
RECORD              /IDANDFT/PTFTS(3)
INTEGER*2   NUMVALUS
INTEGER*2   PTDTYPES(3)
INTEGER*2   NUMOK
RECORD              /ERRBLOCK/ERR

STRUCTURE   /DATAFORMAT/
    INTEGER*2       FIELD1
    INTEGER*4       FIELD2
```

```
    REAL*4    FIELD3
END STRUCTURE

RECORD                /DATAFORMAT/PTDATAS


DTYPES(1) = DTYPSHRT
DTYPES(2) = DTYPLONG
DTYPES(3) = DTYPREAL

CALL MRDBVALS( %VAL(NUMVALUS), %REF(FIELDS),%REF(PTDTYPES),
              %REF(PTDATAS), NUMOK, ERR)
```

WDBOCCS

Writes multiple occurrences of multiple fields to a repeat area in one database record.

Format

```
MWDBOCCS( recid, numfts, ptfts, ptdtypes, frstoc, lastoc,
ptdatas, occsok, ftsok, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the repeat area.

numfts

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numfts* is the number of fields to write to each occurrence.

ptfts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*ptfts* is an array containing the field tags of the fields to write to each occurrence. The occurrence number of each FT should be zero and there should be *numfts* elements in the array.

ptdtypes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*ptdtypes* is an array that identifies the data type of each field to write. Each element in the array is a data type as defined in the **setcim.h** include file. For example, if the fifth field is a 16-bit signed integer, then the fifth element of *ptdtypes* is DTYPSHRT.

frstoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*frstoc* is the starting occurrence number to write.

lastoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*lastoc* is the ending occurrence number to write.

ptdatas

Data Type	array of addresses
Access	Output only
Mechanism	Passed by reference
Description	*ptdatas* is an array of addresses. Each address is the location of an array of values to write to the occurrences of a specific field. The array must contain data for all of the occurrences to write and be of the type specified in *ptdtypes*.

occsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*occsok* is the actual number of occurrences written.

ftsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*ftsok* is the actual number of fields written. Unless there is an error, *ftsok* will equal *numfts*.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
long            recid,          /*  ID of the record                    */
                ptfts[3];               /*  Field Tags in the repeat area    */
short           frstoc,         /*  Starting occurrence to write        */
                lastoc,         /*  Ending occurrence to write          */
                numfts,                 /*  Number of fields to write
        */
                occsok,                 /*  Number of occurrences written  */
                ftsok;          /*  Number of fields written            */
short           ptdtypes[3];    /*  Data types for each value to write  */
short           field1[200];    /*  Destination for short data          */
long            field2[200];    /*  Destination for long data           */
float           field3[200];    /*  Destination for float data          */
short           *ptdatas[3];    /*  Data to write                       */
ERRBLOCK        err;            /*  Error for record                    */
ptdtypes[0] = DTYPSHRT;
ptdtypes[1] = DTYPLONG;
ptdtypes[2] = DTYPREAL;
ptdatas[0] = field1;
ptdatas[1] = field2;
ptdatas[2] = field3;

MWDBOCCS(recid, numfts, ptfts, ptdtypes, frstoc, lastoc, ptdatas, &occsok,
&ftsok,&err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   PTFTS(3)
INTEGER*2   FRSTOC
INTEGER*2   LASTOC
INTEGER*2   NUMFTS
INTEGER*2   PTDTYPES(3)
INTEGER*4   PTDATAS(3)
INTEGER*2   OCCSOK
INTEGER*2   FTSOK
INTEGER*2   FIELD1(200)
INTEGER*4   FIELD2(200)
REAL*4              FIELD3(200)
RECORD              /ERRBLOCK/ERR
PTDTYPES(1) = DTYPSHRT
PTDTYPES(2) = DTYPLONG
PTDTYPES(3) = DTYPREAL
PTDATAS(1) = %LOC(FIELD1)
PTDATAS(2) = %LOC(FIELD2)
PTDATAS(3) = %LOC(FIELD3)

CALLM WDBOCCS(%VAL(RECID),%VAL(NUMFTS),%REF(PTFTS),
      %REF(PTDTYPES), %VAL(FRSTOC),%VAL(LASTOC),%REF(PTDATAS),
      OCCSOK, FTSOK, ERR)
```

MWDBVALS

Writes multiple fields in multiple records in the database.

Format

```
MWDBVALS(numvalus, fields, ptdtypes, ptdatas, numok, error)
```

Arguments

numvalus

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numvalus* is the number of values to be written.

fields

Data Type	IDANDFT array
Access	Input only
Mechanism	Passed by reference
Description	*fields* is an array of record and field IDs to be written.

ptdtypes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*ptdtypes* is the address of an array of the respective data types of the field in *fields* (i.e., *ptdtypes*[1] is the data type of field *fields*[1]).

ptdatas

Data Type	type aligned address
Access	Input only
Mechanism	Passed by reference
Description	*ptdatas* is the address of the destination of the data. Values are consecutive rounded to word boundaries. As the list of fields often refers to different types and sizes of data, *ptdatas* is usually a structure.

numok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numok* is the actual number of values written.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
IDANDFT      fields[3];              /*  List of record & field IDs to be written
*/
short        numvalus,     /*  Number of fields to write          */
             ptdtypes[3],  /*  List of field tags' data types     */
             numok;                  /*  Number of values successfully written
*/
 ERRBLOCK    err;          /*  Error status */


 struct {    short field1;           /*  Data to write */
             long  field2;
             float field3;
        } ptdatas;
ptdtypes[0] = DTYPSHRT;
ptdtypes[1] = DTYPLONG;
ptdtypes[2] = DTYPREAL;

MWDBVALS(numvalus, fields, ptdtypes, &ptdatas, &numok, &err);
```

Sample FORTRAN Program

```
RECORD       /IDANDFT/PTFTS(3)
INTEGER*2    NUMVALUS
INTEGER*2    PTDTYPES(3)
INTEGER*2    NUMOK
RECORD       /ERRBLOCK/ERR
STRUCTURE    /DATAFORMAT/
   INTEGER*2FIELD1
   INTEGER*4FIELD2
```

```
   REAL*4      FIELD3
END STRUCTURE
RECORD                /DATAFORMAT/PTDATAS

DTYPES(1) = DTYPSHRT
DTYPES(2) = DTYPLONG
TYPES(3) = DTYPREAL
CALL MWDBVALS ( %VAL(NUMVALUS), %REF(FIELDS),%REF(PTDTYPES),
                        %REF(PTDATAS), NUMOK, ERR)
```

NAMSZDEF

Returns the record name size from a definition record.

Returns

short word

Returns the size of the record name field as defined by the definition record.

Format

```
NAMSZDEF(defid)
```

Arguments

defid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*defid* is the ID of a definition record.

Sample C Program

```
long        defid;      /*  Definition record ID      */
short       size;       /*  Size of record name field*/

size = NAMSZDEF(defid);
```

Sample FORTRAN Program

```
INTEGER*4    DEFID
INTEGER*2    SIZE

SIZE = NAMSZDEF( %VAL(DEFID))
```

NXTREFER

Finds the next reference to a given record and optionally to a given field.

Format

```
NXTREFER(idcheck, ftcheck, ptiduse, ptftuse)
```

Arguments

idcheck

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*idcheck* is the ID of the record to be checked.

ftcheck

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ftcheck* is the field tag to be checked. If *ftcheck* is equal to 0, search for all references to *idcheck* including definition records. If *ftcheck* is not equal to 0, find all references to the field "*idcheck ftcheck*".

ptiduse

Data Type	long word
Access	Input/Output
Mechanism	Passed by reference
Description	On entry - if *ptiduse* is equal to 0, start searching at the beginning of the database. If *ptiduse* is not equal to 0, start searching at the field after field "*ptiduse ptftuse*". On exit, the following is true:

PTIDUSE	PTFTUSE	Meaning
Nonzero	0	*ptiduse* is a record defined by *idcheck.*
Nonzero	Nonzero	*ptiduse* is a record that references *idcheck*. The field in *ptiduse* that contains the reference is *ptftuse*.
0	Irrelevant	Search is complete.

ptftuse

Data Type	long word
Access	Input/Output
Mechanism	Passed by reference
Description	On exit, *ptftuse* is the field tag of the field in record *ptiduse* that references *idcheck*.

Sample C Program

```
long    idcheck,      /*  Record ID of record to check              */
        ftcheck,      /*  Field tag of field to check                */
        ptiduse,      /*  Starting and returned reference record ID's */
        ptftuse;      /*  Starting and returned reference field tags  */

NXTREFER (idcheck, ftcheck, &ptiduse, &ptftuse);
```

Sample FORTRAN Program

```
INTEGER*4   IDCHECK
INTEGER*4   FTCHECK
INTEGER*4   PTIDUSE
INTEGER*4   PTFTUSE

CALL NXTREFER ( %VAL(IDCHECK), %VAL(FTCHECK), PTIDUSE, PTFTUSE)
```

R2ASCIIDB

Converts a real value to ASCII in the format specified by a database field.

Format

```
R2ASCIIDB(recid, ft, realdata, ptbuff, maxchars, numchars, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID for the field tag.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field defining the real format.

realdata

Data Type	single/double precision real
Access	Input only
Mechanism	Passed by reference
Description	*realdata* is the real value to be converted and is of the same precision (single or double) as that defined by the field in the record.

ptbuff

Data Type	character array
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* is the address of the buffer to receive the ASCII data.

m**axchars**

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*maxchars* specifies the maximum number of characters in the buffer.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the number of characters which are normally written to a buffer. If *numchars* is greater than *maxchars*, only *maxchars* characters are used.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,        /*  Record ID                  */
            ft;           /*  Field tag                  */
float       realdata;           /*  Real value                 */
char        ptbuff[100];  /*  Address of buffer to receive the       */
                          /*  ASCII data                 */
short       maxchars,     /*  Max number of chars in the buffer      */
            numchars;     /*  # of chars normally written to buffer  */
ERRBLOCK    err;

maxchars = 100;
R2ASCIIDB (recid, ft, &realdata, ptbuff, maxchars, &numchars, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
REAL*4               REALDATA
CHARACTER*100        PTBUFF
INTEGER*2    MAXCHARS
INTEGER*2    NUMCHARS
RECORD               /ERRBLOCK/ERR

CALL R2ASCIIDB( %VAL(RECID), %VAL(FT), REALDATA, %REF(PTBUFF),
                %VAL(MAXCHARS), NUMCHARS, ERR)
```

RDBASCII

Reads from a database field and converts it to ASCII. The ASCII conversion matches that defined by the field's display format.

Format

```
RDBASCII(recid, ft, ptbuff, maxchars, numchars, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record containing the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field containing the data.

ptbuff

Data Type	character array
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* is the buffer to receive the ASCII data.

maxchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*maxchars* specifies the maximum number of ASCII characters in *ptbuff* to use.

numchars

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numchars* is the number of characters that are required to hold the data. If *numchars* is greater then *maxchars*, only *maxchars* characters are used. If there is no format record defined for the field, then *numchars* will always be equal to *maxchars* + 1.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,              /* Record ID of the record containing the data
        */
            ft;         /* Field tag of the field containing the data        */
char        ptbuff[10];         /* ASCII data                                */
short       maxchars,/* Max number of ASCII characters to use */
            numchars;/* Number of ASCII characters used                   */
ERRBLOCK  err;
.
RDBASCII (recid, ft, ptbuff, maxchars, &numchars, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
CHARACTER*10       PTBUFF
INTEGER*2    MAXCHARS
INTEGER*2    NUMCHARS
RECORD               /ERRBLOCK/ERR

CALL RDBASCII(%VAL(RECID),%VAL(FT),%REF(PTBUFF),
                %VAL(MAXCHARS),NUMCHARS, ERR)
```

RDBOCCS

Reads multiple occurrences of a single field from one record in the database.

Format

```
RDBOCCS(recid, ft, frstoc, lastoc, datatype, ptdatas, numok,
error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the Record ID of the record being accessed.

f**t**

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag in a repeat area with its occurrence number set to 0.

frstoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*frstoc* is the starting occurrence number to be read.

lastoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*lastoc* is the ending occurrence number to be read.

datatype

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*datatype* is the data type of the field as defined in the **setcim.h** include file.

ptdatas

Data Type	type aligned address
Access	Output only
Mechanism	Passed by reference
Description	*ptdatas* is the address of the destination of the data. Values are consecutively rounded to word boundaries.

Note: *ptdatas* is a buffer of values with each data type aligned on its natural boundary. For example, bytes are aligned on byte boundaries, long words on long words boundaries.

numok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numok* is the actual number of occurrences read.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,          /* ID of the record                    */
            ft;             /* Field Tag in the repeat area        */
short       frstoc,         /* Starting occurrence to read              */
            lastoc,         /* Ending occurrence to read                */
            datatype,               /* Datatype of field tag            */
            ptdatas[200], /* Destination of data read          */
            numok;                  /* Number of occurrences read       */
ERRBLOCK    err;

RDBOCCS  (recid, ft, frstoc, lastoc, datatype, ptdatas, &numok, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*2   FRSTOC
INTEGER*2   LASTOC
INTEGER*2   DATATYPE
INTEGER*2   PTDATAS(200)
INTEGER*2   NUMOK
RECORD              /ERRBLOCK/ERR

CALL RDBOCCS( %VAL(RECID), %VAL(FT), %VAL(FRSTOC),%VAL(LASTOC),
      %VAL(DATATYPE), %REF(PTDATAS), NUMOK, ERR)
```

RDBVALS

Reads multiple values from one record in the database.

Format

```
RDBVALS(recid, numvalus, ptfts, ptdtypes, ptdatas, numok, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record being accessed.

numvalus

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numvalus* is the number of values to be read.

ptfts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*ptfts* is the address of an array of field tags to be read.

ptdtypes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*ptdtypes* is the address of an array of the respective data types of the field tags in *ptfts* (i.e., *ptdtypes*[1] is the data type of field *ptfts*[1]).

ptdatas

Data Type	type aligned address
Access	Output only
Mechanism	Passed by reference
Description	*ptdatas* is the address of the destination of the data. Values are consecutive rounded to word boundaries. As the list of fields often refers to different types and sizes of data, *ptdatas* is usually a structure.

numok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numok* is the actual number of values read.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
long    recid,          /* ID of the record             */
        ptfts[3];               /* List of field tags to be read        */
short   numvalus,       /* Number of tags to read    */

/* List of field tag's data types */

ptdtypes[] =  { DTYPSHRT, DTYPLONG, DTYPREAL },
        numok;                  /* Number of values successfully read */
 struct      {     short field1; /*  Destination of data */
              long  field2;
              float field3;
        } ptdatas;

ERRBLOCK *err;

RDBVALS(recid, numvalus, ptfts, ptdtypes, &ptdatas, &numok, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    PTFTS(3)
INTEGER*2    NUMVALUS
INTEGER*2    PTDTYPES(3)
INTEGER*2    NUMOK
RECORD               /ERRBLOCK/ERR
STRUCTURE   /DATAFORMAT/
  INTEGER*2 FIELD1
```

```
   INTEGER*4 FIELD2
   REAL*4            FIELD3
END STRUCTURE
RECORD               /DATAFORMAT/PTDATAS

DTYPES(1) = DTYPSHRT
DTYPES(2) = DTYPLONG
DTYPES(3) = DTYPREAL

CALL RDBVALS(%VAL(RECID),%VAL(NUMVALUS),%REF(PTFTS),
        %REF(PTDTYPES) %REF(PTDATAS), NUMOK, ERR)
```

REALADD2DB

Writes a real number to the database (data type = DTYPREAL). This routine is designed to accommodate FORTRAN calling conventions.

Format

```
REALADD2DB(recid, ft, realdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record to contain the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field to contain the data.

realdata

Data Type	single precision real
Access	Input only
Mechanism	Passed by reference
Description	*realdata* is the real value to be written to the database.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* returns an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,              /* Record ID of the record to contain the data */
            ft;         /* Field tag of the field to contain the  data      */
float       realdata;   /*  Real value                          */
ERRBLOCK err;

REALADD2DB (recid, ft, &realdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
REAL*4              REALDATA
RECORD              /ERRBLOCK/ERR

CALL REALADD2DB ( %VAL(RECID), %VAL(FT), REALDATA, ERR)
```

RECIDMAX

Obtains the highest usable record ID in the database.

Format

```
RECIDMAX()
```

Returns

long word

RECIDMAX returns the highest record ID in the database.

Sample C Program

```
long    maxid;          /* Returned max record ID in database      */
maxid = RECIDMAX();
```

Sample FORTRAN Program

```
INTEGER*4    MAXID
MAXID = RECIDMAX()
```

RECTYPEOK

Returns TRUE if the specified record is of a given type.

Returns

integer

Returns a nonzero value if the record is of the given type or zero if it is not.

Format

```
RECTYPEOK (recid, rectype)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record.

rectype

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*rectype* is the record ID of a definition record or a record type identifier from the table below:

Record Type	Identifier
Any Defined Record	RTYPANYRECORD
External Task Record	RTYPEXTASK
Field Name Record	RTYPFLDNAME
Definition Record	RTYPDEFINE
Select Descriptor Record	RTYPSELECT
Disk History Record	RTYPDSKHIST
History Summary Line Record	RTYPHSUMLIN
Pseudo Summary Line Record	RTYPPSUMLIN
Normal Summary Line Record	RTYPPSUMLIN
Integer Format Record	RTYPIFORMAT
Real Format Record	RTYPRFORMAT
Timestamp Format Record	RTYPTFORMAT
Detail Display Record	RTYPDETDSPLY
External Task Record Definition Record	RTYPDEFEXTASK
Field Name Record Definition Record	RTYPDEFFLDNAME
Definition Record Definition Record	RTYPDEFDEFINE
Select Descriptor Record Definition Record	RTYPDEFSELECT
Disk History Record Definition Record	RTYPDEFDSKHIST
Folder Record	RTYPFOLDER
Summary Record	RTYPSUMMARY
Integer Validation Processing	RTYPIVALID
Real Validation Processing	RTYPRVALID
Data Compression Processing	RTYPBOXCAR

Sample C Program

```
long        recid,        /*  Record ID          */
            rectype;            /*  Data type == RTYP?     */
int         recokay;            /*  For the result         */

rectype = RTYPEXTASK;
okay    = RECTYPEOK (recid, rectype);
```

Sample FORTRAN Program

```
INTEGER*4    RECID

INTEGER*4    RECTYPE
INTEGER*4    RECOKAY

RECTYPE  = RTYPEXTASK
RECOKAY = RECTYPEOK (%VAL(RECID), %VAL(RECTYPE))
```

REID2DB

Writes a record ID to the database (data type = DTYREID).

Format

```
REID2DB(recid, ft, iddata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field in which to write.

iddata

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*iddata* is the buffer containing the record ID value.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /*  Record ID      */
            ft,         /*  Field Tag      */
            iddata;     /*  Output buffer for ID  */
ERRBLOCK    err;

REID2DB (recid, ft, iddata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*4   IDDATA
RECORD          /ERRBLOCK/ERR

CALL REID2DB ( %VAL(RECID), %VAL(FT), %VAL(IDDATA), ERR)
```

RESTORSNAP

Reads a database snapshot disk file into memory.

Warning: This routine overwrites the current database.

Format

```
RESTORSNAP(ptfname, numchars, blksin, wordsin, error)
```

Arguments

ptfname

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptfname* is the address of the buffer containing the filename.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* is the number of characters in the buffer.

blksin

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*blksin* is the number of disk blocks to read.

wordsin

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*wordsin* contains the number of words in the database read from disk.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
char    ptfname[40]; /*  Address of the buffer containing the file name
        */
short   numchars;    /*  # of characters in the buffer              */
long    blksin,
        wordsin;             /*  # of words in the database read from disk
        */
ERRBLOCK err;

RESTORSNAP (ptfname, numchars, &blksin, &wordsin, &err);
```

Sample FORTRAN Program

```
CHARACTER*40      PTFNAME
INTEGER*2   NUMCHARS
INTEGER*4   BLKSIN
INTEGER*4   WORDSIN
RECORD            /ERRBLOCK/ERR

CALL RESTORSNAP ( %REF(PTFNAME), %VAL(NUMCHARS),
       BLKSIN, WORDSIN, ERR)
```

RHIS21AGGREG

Generates statistics for each time interval between two specified times.

Format

```
void RHIS21AGGREG( timeweight, step, recid, ft, ptTimeOld,
ptTimeNew, ptInterval, timealign, dsadjust, maxperiods,
numtimecodes, numdoublecodes, numshortcodes, timecodes,
doublecodes, shortcodes, timevalues, doublevalues, shortvalues,
numperiods, err);
```

Arguments

timeweight

Data Type	integer
Access	Input only
Mechanism	Passed by value
Description	*Timeweight* is a flag that indicates if statistics should be calculated using the best fit method (time weighted) or not. If 1, then statistics are calculated with the best fit method. The time period containing the last value and those past the latest value are not calculated. If 2, then statistics are calculated with the best fit method. The time period containing the latest value is calculated, but none past this period are calculated. If 3, then statistics are calculated with the best fit method. All time periods are calculated, even those past the latest value. If 0, then the function will perform statistics using actual values. If –1, use actual values and include the period containing the latest value. If –2, use actual values and include all periods, even those starting after the latest value.

step

Data Type	Integer	
Access	Input only	
Mechanism	Passed by value	
Description	*step* values are:	
	Value	Meaning
	0	no
	1	yes

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record that contains the history repeat area.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of a field in the history repeat area (which must have a valid occurrence number, such as 1) or repeat area count field tag (which must have an occurrence number of zero).

ptTimeOld

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	An extended timestamp. *PtTimeOld* specifies the start time for occurrences to read from history. The time is in GMT. This function will not adjust the daylight saving time. If the *ptTimeOld* is older than the allowable time (i.e., older than the time the record was created, the oldest allowable time is used instead of the user's requested time.

ptTimeNew

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	An extended timestamp. *PtTimeNew* specifies the end time for occurrences to read from history. The time is in GMT. This function will not adjust the daylight saving time.

ptInterval

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	*ptInterval* points to an XUSTS structure containing the time difference between the start of consecutive periods, specified as seconds and microseconds. If *dsadjust* = 1, then this difference must be at least two hours.

timealign

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	*timealign* indicates the time used to align the start of periods. If *timealign* = 0, then no alignment is done and the start of the first (oldest) period is **ptTimeOld*. If *timealign* > 0, then the periods are made earlier (if necessary) by the minimum amount to make the periods start at an integral multiple of **ptInterval* before or after the "alignment time" (so the first (oldest) period always contains **ptTimeOld*). If *timealign* = 1, then the "alignment time" is the "start of day" (which also indicates the "start of week"). If *timealign* = 2, then the "alignment time" is the "start of shift." For example, if *timealign* = 1 when obtaining daily statistics, then all periods begin at the same time in the day as the "start of day." The "start of day" and the "start of shift" are specified in the database. The "start of day" is specified in the AGGREG_DAY_START and AGGREG_SHIFT_START fields of the appropriate history parameters records (i.e., HistoryParameters).

dsadjust

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	*dsadjust* indicates if adjustments for Daylight Savings Time changes are made. If *dsadjust* = 0, then no adjustments are made. All time calculations are made with an absolute interval size. If *dsadjust* = 1, then adjustments for time changes are made. *dsadjust* can only be set to 1 if *ptInterval* is greater than or equal to two hours; otherwise an error will be returned. Two time adjustments are made: If an alignment time is specified in the *timealign* parameter and the stored alignment time and *ptTimeOld* do not occur in the same time standard period, then the start time of the first interval will be adjusted to match the alignment time in local time. For example, if the alignment time is specified as 06:00:00 on a day in standard time and *ptTimeOld* is 12:00:00 on a day in Daylight Savings Time, then the start time of the first daily interval calculated will be at 06:00:00 local time on the same day for which *ptTimeOld* is specified. Without the adjustment, the first daily interval would begin at 07:00:00 local time. The second time adjustment will occur if the time span between *ptTimeOld* and *ptTimeNew* crosses the boundary of a time change. The aggregate interval that actually crosses the time change boundary will be shortened by one hour if the time change is from standard to DST. The interval will be lengthened by one hour if the change is from DST to standard. All other intervals in the time span will remain at the length specified in *ptInterval*. This adjustment will keep interval start times within the same local time boundaries. For example, if daily aggregates are specified to start at 08:00:00, then all intervals will start at 08:00:00 local time. Without the adjustment, all intervals would be 24 hours, and if a time change boundary is crossed, the intervals after the time change would start at 07:00:00 or 09:00:00 local time, depending on the boundary crossed.

maxperiods

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*maxperiods* is the maximum number of periods. The output arrays should be at least this big.

numtimecodes

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*numtimecodes* is the number of time values to be returned for each period. This is the number of elements in the *timecodes* array. Typically zero to four since there are four different kinds of timestamp aggregate data that can be returned.

numdoublecodes

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*numdoublecodes* is the number double precision real values to be returned for each period. This is the number of elements in the *doublecodes* array. Typically zero to nine since there are nine different kinds of double-precision aggregate data that can be returned.

numshortcodes

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*numshortcodes* is the number of short integer values to be returned for each value. This is the number of elements in the *shortcodes* array. Typically zero to one since there is only one kind of short integer aggregate data that can be returned.

timecodes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	Each element of *timecodes* indicate a kind of timestamp value to be returned for each period. The possible timecodes are defined in the **setcim.h** file. • AG_TIME_START defines a timecode for the time at beginning of period. • AG_TIME_MIDDLE defines a timecode for the time at middle of period. • AG_TIME_END defines a timecode for the time at end of period. • AG_TIME_AFTER defines a timecode for the time after end of period (start of next period). • AG_TIME_MIN defines a timecode for the minimum time value within an aggregate period. • AG_TIME_MAX defines a timecode for the maximum time alue within an aggregate period.

Doublecodes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	Each element of *doublecodes* indicate a kind of double-precision value to be returned. The possible doublecodes are defined in the **setcim.h** file. • AG_DBL_GOOD defines a doublecode for the number of good values within each period. If *timeweight* is 1, then the amount of time values that were good during the period is returned. • AG_DBL_NG defines a doublecode for the number of values that are not good within each period. If *timeweight* is 1, then the amount of time values that were not good during the period is returned. • AG_DBL_MIN defines a doublecode for the minimum of the good values within each period. • AG_DBL_MAX defines a doublecode for the maximum of the good values within each period. • AG_DBL_RNG defines a doublecode for the range of the good values within each period. • AG_DBL_SUM defines a doublecode for the sum of the good values within each period. If *timeweight* is 1, then the time-weighted sum of good values is returned. This will result in a very large value since the values have been multiplied by time to arrive at this sum. • AG_DBL_AVG defines a double code for the average of the good values within each period. If *timeweight* is 1, then the time-weighted average of the good values is returned. • AG_DBL_VAR defines a double code for the variance of the good values within each period. If *timeweight* is 1, then the time-weighted variance of the good values is returned. • AG_DBL_STD defines a double code for the standard deviation of the good values within each period. If *timeweight* is 1, then the time-weighted standard deviation of the good values is returned.

shortcodes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	Each element of *shortcodes* indicate a kind of short integer value to be returned. For example, AG_SHRT_QLEVEL is a shortcode of the period's the quality "level". The possible shortcodes are defined in the **setcim.h** file.

timevalues

Data Type	XUSTS array
Access	Output only
Mechanism	Passed by reference
Description	The *timevalues* array should be sized to hold *numtimecodes* times *maxperiods* elements.

doublevalues

Data Type	double precision array
Access	Output only
Mechanism	Passed by reference
Description	The *doublevalues* array should be sized to hold *doublecodes* times *maxperiods* elements.

shortvalues

Data Type	short word array
Access	Output only
Mechanism	Passed by reference
Description	The *shortvalues* array should be sized to hold *shortcodes* times *maxperiods* elements.

numperiods

Data Type	long integer
Access	Output only
Mechanism	Passed by reference
Description	*numperiods* returns the number of periods for which aggregates were calculated.

ptError

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*ptError* returns error information as defined in the **setcim.h** include file.

Each period (after the first) starts *ptInterval after the previous period starts. Each period ends 1 microsecond before the next period starts. All periods must start before *ptTimeNew. All periods must end on or before the time of the newest value (in history or a "current" value). *numperiods is set to the number of periods for which aggregate statistics are calculated, and will be the largest number <= maxperiods, which is consistent with these conditions. For example, if timealign = 0, *ptInterval specifies +2:00:00, maxperiods = 7, *ptTimeOld specifies 1-jan-98 4:00:00, *ptTimeNew specifies 1-JAN-98 6:00:01, and there is a value with a time of 1-JAN-98 8:00:00, then the first period would be 1-JAN-98 4:00:00 through 1-JAN-98 5:59:59.999999, the second period would be 1-JAN-98 6:00:00 through 1-JAN-98 7:59:59.999999, and *numperiods would be set to 2. However, *numperiods would be set to 1 if maxperiods = 1 or if *ptTimeNew specifies 1-JAN-98 6:00:00 or if the newest value has a time of 1-JAN-98 7:59:59.

Using the example in the previous paragraph, if numtimes = 1, timecodes[0] specifies the middle of the period, numdoubles = 2, doublecodes[0] specifies average, doublecodes[1] specifies number of good values, and numshorts == 0, then timevalues[0] would be set to 1-JAN-98 5:00:00, doublevalues[0] would be set to the average for the first period, doublevalues[1] would be set to the number of good values for the first period, and timevalues[1], doublevalues[2], and doublevalues[3] would be set to similar values for the second period.

Sample C Program

```
////////////////////////////////////////////////////
#define NUMTIMECODES 4
#define NUMDOUBLECODES 9
#define NUMSHORTCODES 1

ERRARRAY    errar;
long        recid;
long        ft = 0x24190000;
short       errsz;
int timeweight = 0;
int stepped = 0;
XUSTS timeold;
XUSTS timenew;
XUSTS interval;
int timealign = 0;
int dsadjust = 0;
long maxperiods = 1000;
```

```
short timecodes[NUMTIMECODES] =     {AG_TIME_START,
                                AG_TIME_MIDDLE,
                                AG_TIME_END,
                                AG_TIME_AFTER};

short doublecodes[NUMDOUBLECODES] =  {AG_DBL_GOOD,
                                AG_DBL_NG,
                                AG_DBL_MIN,
                                AG_DBL_MAX,
                                AG_DBL_RNG,
                                AG_DBL_SUM,
                                AG_DBL_AVG,
                                AG_DBL_VAR,
                                AG_DBL_STD};

short shortcodes[NUMSHORTCODES] = {AG_SHRT_QLEVEL};

XUSTS *ptTimevalues;
double *ptDoublevalues;
short *ptShortvalues;
long numperiods;
ERRBLOCK err;
short datatype;
.
.
.
// Allocate memory to hold results
ptTimevalues = malloc(maxperiods * NUMTIMECODES * sizeof(XUSTS));
ptDoublevalues = malloc(maxperiods * NUMDOUBLECODES * sizeof(double));
ptShortvalues = malloc(maxperiods * NUMSHORTCODES * sizeof(short));


// Call RHIS21AGGREG
RHIS21AGGREG(timeweight, stepped, recid, ft, &timeold, &timenew,
          &interval, timealign, dsadjust, maxperiods, NUMTIMECODES,
```

```
        NUMDOUBLECODES, NUMSHORTCODES, timecodes, doublecodes,
        shortcodes, ptTimevalues, ptDoublevalues, ptShortvalues,
        &numperiods, &err);
if (err.ERRCODE != SUCCESS)
{
    ERRMESS (&err, errar, &errsz);
    printf("**RHIS21AGGREG Error: %.*s**\n", errsz, errar);
    return;
}
```

Sample FORTRAN Program

```
PARAMETER   (MAX_PERIODS = 100)
INTEGER*4   TIMEWEIGHT
INTEGER*4   RECID
INTEGER*4   FT
RECORD/XUSTS/     TIMEOLD
RECORD/XUSTS/     TIMENEW
RECORD/XUSTS/     INTERVAL
INTEGER*4   TIMEALIGN
INTEGER*4   MAXPERIODS
INTEGER*4   NUMTIMECODES
INTEGER*4   NUMDOUBLECODES
INTEGER*4   NUMSHORTCODES
INTEGER*2   TIMECODES(3) /     AG_TIME_START,
                               AG_TIME_MIDDLE,
                               AG_TIME_END,
                               AG_TIME_AFTER /
INTEGER*2   DOUBLECODES(9)  / AG_DBL_GOOD,
                               AG_DBL_NG,
                               AG_DBL_ MIN,
                               AG_DBL_MAX,
                               AG_DBL_RNG,
                               AG_DBL_SUM,
                               AG_DBL_AVG,
```

```
                                AG_DBL_VAR,
                                AG_DBL_STD /
INTEGER*2    SHORTCODES(1)  / AG_SHRT_QLEVEL /
RECORD/XUSTS/      TIME(4,MAX_PERIODS)
REAL*8             DOUBLEVALUES(9,MAX_PERIODS)
INTEGER*2    SHORTVALUES(MAX_PERIODS)
INTEGER*4    NUMPERIODS
RECORD/ERRBLOCK/ ERR

FT = 0X24190001
MAXPERIODS = MAX_PERIODS
NUMTIMECODES = 4
NUMDOUBLECODES = 9
NUMSHORTCODES = 1

RHIS21AGGREG (%VAL(TIMEWEIGHT),%VAL(0),VAL(RECID),%VAL(FT),
      %REF(TIMEOLD),
%REF(TIMENEW),%REF(INTERVAL),%VAL(TIMEALIGN),      %VAL(0),
%VAL(MAXPERIODS), %VAL(NUMTIMECODES),
      %VAL(NUMDOUBLECODES),%VAL(NUMSHORTCODES),
      %REF(TIMECODES), %REF(DOUBLECODES), %REF(SHORTCODES),
      %REF(TIMEVALUES), %REF(DOUBLEVALUES),
      %REF(SHORTVALUES), NUMPERIODS, %REF(ERR))
```

RHIS21DATA

Reads multiple occurrences of multiple historical fields, reading the occurrences in chronological order (older occurrences first), and storing the values read into multiple data arrays. Both the in memory repeat areas as well as the disk-based history repeat areas are read.

This routine, which does **not** use sequence numbers, will not work with non-history repeat areas, history repeat areas without disk history configuration, and archiving on/off fields that are nonzero.

Format

```
RHIS21DATA (mode, step, outsiders, recid, ft, timeold, timenew,
numfts, fts, datatypes, maxoccs, keylevels, keytimes, ptdatas,
ptoccsok, ftsok, error)
```

Arguments

mode

Data Type	integer
Access	Input only
Mechanism	Passed by value
Description	*mode* is the type of the request. Valid values for mode are: • H21_GET_TIMES – Returns interpolated values between *timeold* and *timenew*. The time interval is (*newtime*–*oldtime*)/*maxoccs* • H21_GET_TIMES2 – Returns interpolated values between *timeold* and *timenew*. The time interval is (*newtime*–*oldtime*)/(*maxoccs*–1) • H21_GET_BEST_FIT – Interpolates both end points, divide the time between *timeold* and *timenew* into intervals and for each interval returns lowest or highest value in the interval. • H21_GET_ACTUALS – Returns actual values as stored in history. • H21_GET_ACTUALS_AND_CURRENT – Returns actual values as stored in history as well as the current value stored in the fixed area of the record. • H21_GET_TIMES_EXTENDED – Is like H21_GET_TIMES except if the **step** argument is set and *timenew* is newer than the newest value in history, that newest value will be repeated in all intervals between the time of the newest value and *timenew.* • H21_GET_TIMES2_EXTENDED – Is like H21_GET_TIMES2 except if the **step** argument is set and *timenew* is newer than the newest value in history, that newest value will be repeated in all intervals between the time of the newest value and *timenew.*

Step

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	*step* is the discrete flag. If it is TRUE, this routine returns extra values so that the returned data set can be used to create step plots.

outsiders

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	If non-zero, get the before and after occurrences. Mode must be set to H21_GET_ACTUALS if outsiders is set to non-zero. In other words, non-zero values in outsiders are only valid when mode is set to H21_GET_ACTUALS. RHIS21DATA will return an error code of DHREADER (-68) along with the value 12 in the ERR2 field of the error block if outsiders is non-zero and mode is not H21_GET_ACTUALS.

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record that contains the history repeat area.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of a field in the history repeat area (which must have a valid occurrence number, such as 1) or repeat area count field tag (which must have an occurrence number of zero).

timeold

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	*timeold* is an extended microsecond timestamp. *timeold* is the start time for occurrences to read from history.

timenew

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	*timenew* is an extended microsecond timestamp. *timenew* is the end time for occurrences to read from history.

numfts

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numfts* is the number of fields to be read from each occurrence. *numfts* is the number of array elements in *fts*, *datats*, and *ptdatas*.

fts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*fts* is an array of the field tags of the fields to be read. The occurrence number part of each field tag must be 0.

datatypes

Data Type:	short word array
Access:	Input only
Mechanism:	Passed by reference
Description:	*datatypes* is an array of the data types of the fields to be read.

maxoccs

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*maxoccs* is the maximum number of occurrences to read. *maxoccs* is the number of array elements in the data arrays pointed to by elements of *ptdatas*.

keylevels

Data Type	short word array
Access	Output
Mechanism	Passed by reference
Description	*keylevels* array holds the key quality levels for the occurrences read. This parameter can be set to NULL, in which case key levels for the historical occurrences are not returned.

keytimes

Data Type	XUSTS array
Access	Output
Mechanism	Passed by reference
Description	*keytimes* array holds the key times for the occurrences read. This parameter can be set to NULL, in which case key times for the historical occurrences are not returned.

ptdatas

Data Type	pointer array
Access	Output only (for de-referenced pointers)
Mechanism	Passed by reference
Description	*ptdatas* is an array of pointers to data arrays, one pointer for each field to be read. Each data array (of the type specified by its datats element) contains the data read from its field in the occurrences (with data for older occurrences first). Only *occsok* elements of each data array are used. If the elements of a data array are byte arrays with an odd number of bytes, then one byte is skipped after each of those byte arrays.

occsok

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*occsok* returns the number of occurrences read.

ftsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*ftsok* returns the number of fields read. *ftsok* returns *numfts* unless there is an error.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* returns an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        id;
long        fts[2] = { 0x241A0000, 0x24190000 };
short       data_ts[2] = { DTYPXTIM, DTYPDUBL };
XTSBLOCK  times[MAXOCCS];
double      vals[MAXOCCS];
XUSTS         keytimes[MAXOCCS];
short       keylevels[MAXOCCS];
void        *ptdatas[2] = { (void *)times, (void *)vals };
XUSTS       xusts_old;
XUSTS       xusts_new;
long        occs_ok;
```

```
short       fts_ok;
ERRBLOCK  err;
RHIS21DATA ( H21_GET_ACTUALS, 0,  0, id, fts[0] + 1, &xusts_old,
&xusts_new, 2,fts,     data_ts, (long)MAXOCCS,  keylevels, keytimes,
ptdatas,&occs_ok,&fts_ok,&err);
```

Sample FORTRAN Program

```
INTEGER*2      FTS_OK
INTEGER*2      DATA_TS(2)
INTEGER*2      KEYLEVELS(MAXOCCS)
INTEGER*4      ID
INTEGER*4      FTS(2)
INTEGER*4      PTDATAS(2)
INTEGER*4      OCCS_OK
REAL*8          VALS(MAXOCCS)

RECORD/XTSBLOCK/ TIMES(MAXOCCS)
RECORD/XUSTS/ KEYTIMES(MAXOCCS)
RECORD/XUSTS/ XUSTS_OLD
RECORD/XUSTS/ XUSTS_NEW
RECORD/ERRBLOCK/ ERR

PTDATAS(1) = %LOC(TIMES(1))
PTDATAS(2) = %LOC(VALS(1))

CALL RHIS21DATA ( %VAL(H21_GET_ACTUALS), %VAL(0), %VAL(0),
%VAL(ID),
        %VAL(FTS(1) + 1), XUSTS_OLD, XUSTS_NEW, %VAL(2), %REF(FTS),
        %REF(DATA_TS), %VAL(MAXOCCS), KEYLEVELS, KEYTIMES,
        %REF(PTDATAS), OCCS_OK, FTS_OK, ERR )
```

RHIS21REV

Reads multiple occurrences of multiple historical fields, reading the occurrences in reverse chronological order (newer occurrences first), and

storing the values read into multiple data arrays. Both the in memory repeat areas as well as the disk-based history repeat areas are read.

This routine, which does **not** use sequence numbers, will not work with non-history repeat areas. Memory occurrences are read from history repeat areas with archiving on/off fields that are zero or that are defined without disk history configuration.

Format

```
RHIS21REV (spare1, spare2, spare3, spare4, recid, ft, timenew,
timeold, numfts, fts, datatypes, maxoccs, keylevels, keytimes,
ptdatas, ptoccsok, ftsok, error)
```

Arguments

spare1

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	Not currently used. Should be zero.

spare2

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	Not currently used. Should be zero.

spare3

Data Type	Integer
Access	Input only
Mechanism	Passed by value
Description	Not currently used. Should be zero.

spare4

Data Type	Integer
Access	Input only
Mechanism	Passed by value

Description	Not currently used. Should be zero.

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record that contains the history repeat area.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of a field in the history repeat area (which must have a valid occurrence number, such as 1) or repeat area count field tag (which must have an occurrence number of zero).

timenew

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	*timenew* is an extended microsecond timestamp. *timenew* is the start time for occurrences to read from history.

timeold

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	*timeold* is an extended microsecond timestamp. *timeold* is the end time for occurrences to read from history.

numfts

Data Type	short word
Access	Input only

Mechanism	Passed by value
Description	*numfts* is the number of fields to be read from each occurrence. *numfts* is the number of array elements in *fts*, *datats*, and *ptdatas*.

fts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*fts* is an array of the field tags of the fields to be read. The occurrence number part of each field tag must be 0.

datatypes

Data Type**:**	short word array
Access:	Input only
Mechanism:	Passed by reference
Description:	*datatypes* is an array of the data types of the fields to be read.

maxoccs

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*maxoccs* is the maximum number of occurrences to read. *maxoccs* is the number of array elements in the data arrays pointed to by elements of *ptdatas*.

keylevels

Data Type	short word array
Access	Output
Mechanism	Passed by reference
Description	*keylevels* array holds the key quality levels for the occurrences read. This parameter can be set to NULL, in which case key levels for the historical occurrences are not returned.

keytimes

Data Type	XUSTS array
Access	Output
Mechanism	Passed by reference
Description	*keytimes* array holds the key times for the occurrences read. This parameter can be set to NULL, in which case key times for the historical occurrences are not returned.

ptdatas

Data Type	pointer array
Access	Output only (for de-referenced pointers)
Mechanism	Passed by reference
Description	*ptdatas* is an array of pointers to data arrays, one pointer for each field to be read. Each data array (of the type specified by its datats element) contains the data read from its field in the occurrences (with data for older occurrences first). Only *occsok* elements of each data array are used. If the elements of a data array are byte arrays with an odd number of bytes, then one byte is skipped after each of those byte arrays.

occsok

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*occsok* returns the number of occurrences read.

ftsok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*ftsok* returns the number of fields read. *ftsok* returns *numfts* unless there is an error.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* returns an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        id;
long        fts[2] = { 0x241A0000, 0x24190000 };
short       data_ts[2] = { DTYPXTIM, DTYPDUBL };
XTSBLOCK  times[MAXOCCS];
double      vals[MAXOCCS];
XUSTS         keytimes[MAXOCCS];
short       keylevels[MAXOCCS];
void        *ptdatas[2] = { (void *)times, (void *)vals };
XUSTS       xusts_new;
XUSTS       xusts_old;
long        occs_ok;
short       fts_ok;
ERRBLOCK  err;
RHIS21REV ( 0, 0, 0,  0, id, fts[0] + 1, &xusts_new,  &xusts_old, 2, fts,
data_ts, (long)MAXOCCS,  keylevels, keytimes,  ptdatas, &occs_ok, &fts_ok,
&err);
```

Sample FORTRAN Program

```
INTEGER*2      FTS_OK
INTEGER*2      DATA_TS(2)
INTEGER*2      KEYLEVELS(MAXOCCS)
INTEGER*4      ID
INTEGER*4      FTS(2)
INTEGER*4      PTDATAS(2)
INTEGER*4      OCCS_OK
REAL*8          VALS(MAXOCCS)

RECORD/XTSBLOCK/ TIMES(MAXOCCS)
RECORD/XUSTS/ KEYTIMES(MAXOCCS)
RECORD/XUSTS/ XUSTS_NEW
RECORD/XUSTS/ XUSTS_OLD
RECORD/ERRBLOCK/ ERR

PTDATAS(1) = %LOC(TIMES(1))
PTDATAS(2) = %LOC(VALS(1))

CALL RHIS21REV ( %VAL(0), %VAL(0), %VAL(0), %VAL(0), %VAL(ID),
        %VAL(FTS(1) + 1), XUSTS_NEW, XUSTS_OLD, %VAL(2), %REF(FTS),
        %REF(DATA_TS), %VAL(MAXOCCS), KEYLEVELS, KEYTIMES,
        %REF(PTDATAS), OCCS_OK, FTS_OK, ERR )
```

ROOTFOLDER

Returns the ID of the root folder.

Format

```
ROOTFOLDER()
```

Arguments

None

Sample C Program

```
long rootid;
rootid = ROOTFOLDER();
```

Sample FORTRAN Program

```
Integer*4 RootID;
ROOTID = ROOTFOLDER()
```

SAVESNAP

Writes a database snapshot to a disk file.

Format

```
SAVESNAP(ptfname, numchars, blksout, wordsin, error)
```

Arguments

ptfname

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptfname* is the address of the buffer containing the field name.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of characters in the buffer.

blksout

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*blksout* is the number of disk blocks written to.

wordsin

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*wordsin* refers to the number of words in the database written to disk.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
char     ptfname[40];      /* Address of the buffer                    */
short    numchars; /* Number of characters in the buffer              */
long     blksout,    /* Number of disk blocks written to        */
         wordsin;    /* # of words in the database written to disk    */
ERRBLOCK err;

SAVESNAP (ptfname, numchars, &blksout, &wordsin, &err);
```

Sample FORTRAN Program

```
CHARACTER*40            PTFNAME
INTEGER*2          NUMCHARS
INTEGER*4          BLKSOUT
INTEGER*4          WORDSIN
RECORD                  /ERRBLOCK/ERR

CALL SAVESNAP( %REF(PTFNAME), %VAL(NUMCHARS), BLKSOUT,
```

WORDSIN, ERR)

SHRT2DB

Writes a short integer to the database (data type = DTYPSHRT).

Format

```
SHRT2DB(recid, ft, shrtdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field in which to write.

shrtdata

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*shrtdata* is the buffer containing the integer value.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long            recid,          /* Record ID                    */
                ft;             /* Field Tag                    */
short           shrtdata;               /* Integer data to write to database
        */
ERRBLOCK    err;
SHRT2DB (recid, ft, shrtdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*2   SHRTDATA
RECORD      /ERRBLOCK/ERR
CALL SHRT2DB( %VAL(RECID), %VAL(FT), %VAL(SHRTDATA), ERR)
```

TMST2ASCII

TMST2ASCII converts a timestamp in to the current InfoPlus.21 ASCII date/time format.

Format

```
TMST2ASCII(timestmp, ptbuff, sizebuff, error)
```

Arguments

timestmp

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*timestmp* is the timestamp to format.,

ptbuff

Data Type	character
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* specifies the address of the buffer to receive the ASCII form of the timestamp.

sizebuff

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*sizebuff* specifies the number of characters in the buffer. Valid sizes are 15, 18 and 20. Any other sizes cause error to be set to 1.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*error* is set to 0 if no error. Otherwise error is set to 1.

Sample 'C' Program

```
char    ptbuff[20];             /*Address of the buffer to receive
the ASCII data                          */
short           sizebuff;               /*Size of the buffer                */
long            time_stamp;  /*Time Stamp                */
char            error;          /*Returns TRUE if time_stamp
is invalid                              */

                .
                .
                .
TMST2ASCII(time_stamp, ptbuff, sizebuff, &error);

                .
                .
                .
```

Sample FORTRAN Program

```
CHARACTER*20            PTBUFF
INTEGER*2           SIZEBUFF
INTEGER*4           TIME_STAMP
BYTE                ERROR
                .
                .
                .
CALL TMST2ASCII( %VAL(TIME_STAMP), %REF(PTBUFF),
        %VAL(SIZEBUFF), ERROR)
                .
                .
                .
```

TS2XTS

Converts an Aspen InfoPlus.21 timestamp to the equivalent extended timestamp. The XTSFAST element of the extended timestamp will be the same as the original timestamp.

Format

```
TS2XTS(timestamp, xts)
```

Arguments

timestamp

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*timestamp* is the InfoPlus.21 timestamp to be converted.

xts

Data Type	XTSBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*xts* is the returned extended timestamp.

Sample C Program

```
long        timestamp;   /* Timestamp to be converted          */
XTSBLOCK    xts;         /*  Returned extended timestamp   */

TS2XTS (timestamp, &xts);
```

Sample FORTRAN Program

```
INTEGER*4   TIMESTAMP
RECORD              /XTSBLOCK/XTS

CALL TS2XTS (%VAL(TIMESTAMP), XTS)
```

VALIDUSA

Checks if a record is in a usable state.

Returns

Returns TRUE if the record is usable.

Format

```
VALIDUSA(recid)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record being tested for usability.

Sample C Program

```
long    recid;      /* Record ID tested for usability  */
char    ok;         /* Test result             */
```

```
ok = VALIDUSA(recid);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
BYTE         OK

OK = VALIDUSA( %VAL(RECID))
```

WDBASCII

Converts ASCII input and writes it to a database field. The appropriate conversion is done to match the field's data type and format.

Format

```
WDBASCII(recid, ft, ptbuff, numchars, operator, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the record ID of the record to contain the data.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field to contain the data.

ptbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference

Description	*ptbuff* is the buffer containing the ASCII data.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of ASCII characters in *ptbuff* to use.

operator

Data Type	byte
Access	Input only
Mechanism	Passed by value
Description	If *operator* = TRUE, then the requested change will be done only if the restrictions imposed by OP_CHANGE_REF_FIELD and OP_CHANGE_REF_VALUES (as defined in the definition record of *recid*) are not violated. If *operator* = FALSE, then the change will be done regardless of the above restrictions.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* will return an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,    /*  Record ID of the record to be changed        */
            ft;       /*  Field tag of the field to be changed         */
char        ptbuff[10]; /*  ASCII data                                  */
short       numchars; /*  Number of ASCII characters to use            */
char        operator;       /*  TRUE if operator changeability is to be
checked */
ERRBLOCK    err;

WDBASCII(recid, ft, ptbuff, numchars, operator, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    FT
CHARACTER    PTBUFF(10)
INTEGER*2    NUMCHARS
BYTE         OPERATOR
RECORD             /ERRBLOCK/ERR

CALL WDBASCII ( %VAL(RECID), %VAL(FT), %REF(PTBUFF),
      %VAL(NUMCHARS), %VAL(OPERATOR), ERR)
```

WDBOCCS

Writes values to multiple occurrences of a single field in one record in the database.

Format

```
WDBOCCS(recid, ft, frstoc, lastoc, datatype, ptdatas, numok,
error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the Record ID of the record accessed.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag in a repeat area with its occurrence number set to 0. If the field is a COS field pointer, any activations generated are the same as if each occurrence was written individually.

frstoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*frstoc* is the starting occurrence number to be written.

lastoc

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*lastoc* is the ending occurrence number to be written.

datatype

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*datatype* is the data type of the field as defined in the **setcim.h** include file. For character and scratch fields, the *datatype* is the size of the character or scratch array. If the length of a character array is set to zero, the entire field is filled with spaces.

ptdatas

Data Type	type aligned address
Access	Input only
Mechanism	Passed by reference
Description	*ptdatas* is the address of the source of the data. Values are consecutively rounded to word boundaries.

Note: *ptdatas* is a buffer of values with each data type aligned on its natural boundary. For example, bytes are aligned on byte boundaries, long words on long words boundaries.

numok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numok* is the actual number of occurrences written.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,          /* ID of the record                    */
            ft;             /* Field Tag in the repeat area        */
short       frstoc,         /* Starting occurrence to write        */
            lastoc,         /* Ending occurrence to write          */
            datatype,               /* Datatype of field tag                 */
            ptdatas[200], /* Source of data written              */
            numok;                  /* Number of occurrences written */
ERRBLOCK    err;

WDBOCCS (recid, ft, frstoc, lastoc, datatype, ptdatas, &numok, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
INTEGER*2   FRSTOC
INTEGER*2   LASTOC
INTEGER*2   DATATYPE
INTEGER*2   PTDATAS(200)
INTEGER*2   NUMOK
RECORD              /ERRBLOCK/ERR

CALL WDBOCCS (%VAL(RECID),%VAL(FT),%VAL(FRSTOC),
        %VAL(LASTOC), %VAL(DATATYPE), %REF(PTDATAS),
        NUMOK, ERR)
```

WDBVALS

Writes multiple values to one record in the database.

Format

```
WDBVALS(recid, numvalus, ptfts, ptdtypes, ptdatas, numok, error)
```

Arguments

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record accessed.

numvalus

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numvalus* is the number of values to be written.

ptfts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*ptfts* is the address of an array of field tags to be written. If any of the field tags are COS field pointers, the activations generated are the same as if each COS field were written individually.

ptdtypes

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*ptdtypes* is the address of an array of the respective data types of the field tags in *ptfts* (i.e., *ptdtypes*[1] is the data type of field *ptfts*[1]). For character and scratch fields, the *datatype* is the size of the character or scratch byte array. If the length of a character array is set to zero, the entire field is filled with spaces.

ptdatas

Data Type	type aligned address
Access	Input only
Mechanism	Passed by reference
Description	*ptdatas* is the address of the source of the data. Values are consecutive rounded to word boundaries. As the list of field often refers to different types and sizes of data, *ptdatas* is usually a structure.

numok

Data Type	short word
Access	Output only
Mechanism	Passed by reference
Description	*numok* is the actual number of values written.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the returned error code as defined in the **setcim.h** include file.

Sample C Program

```
long    recid,              /* ID of the record                    */
        ptfts[3];   /* List of field tags to be written    */
short   numvalus, /* Number of tags to write             */
```

```
/* List of field tags' data types                */
ptdtypes[3]  =   { DTYPSHRT, DTYPLONG, DTYPREAL };
short       numok;    /* Number of values successfully written */
ERRBLOCK err;

struct  {   short field1;       /* Source of data                    */
            long  field2;
            float field3;
 } ptdatas;

WDBVALS (recid, numvalus, ptfts, ptdtypes, &ptdatas, &numok, &err);
```

Sample FORTRAN Program

```
INTEGER*4    RECID
INTEGER*4    PTFTS(3)
INTEGER*2    NUMVALUS
INTEGER*2    PTDTYPES(3)
INTEGER*2    NUMOK
RECORD       /ERRBLOCK/ERR
STRUCTURE    /DATAFORMAT/
    INTEGER*2        FIELD1
    INTEGER*4        FIELD2
    REAL*4    FIELD3
END STRUCTURE

RECORD       /DATAFORMAT/PTDATAS

DTYPES(1)  =  DTYPSHRT
DTYPES(2)  =  DTYPLONG
DTYPES(3)  =  DTYPREAL
CALL WDBVALS ( %VAL(RECID), %VAL(NUMVALUS), %REF(PTFTS),
        %REF(PTDTYPES), %REF(PTDATAS), NUMOK, ERR)
```

WHIS21DAT

Writes one occurrence of multiple historical fields.

The history repeat area written may be in memory repeat areas or disk based history. If any of the field tags are COS field pointers and are written to memory, the COS activations generated are the same as if the field values are written individually.

This routine does NOT use sequence numbers.

This routine will not work with non-history repeat areas, history repeat areas without disk history configuration, and archiving on/off fields that are nonzero.

Format

```
WHIS21DAT(mode, recid, ft, numfts, fts, datatypes, ptdatas,
keylevel, keytime, ftsok, error )
```

Arguments

mode

Data Type	integer
Access	Input only
Mechanism	Passed by value
Description	*mode* is the type of the request. Valid values for mode are: WHIS_TYPE_MODIFY – Modify existing occurrences. If the occurrence with the given key timestamp does not exist, returns an error. WHIS_TYPE_ADDNEW – Add new occurrences, (if necessary, key timestamp will be incremented to make them unique). WHIS_TYPE_UPDATE – If the occurrence with the given key timestamp exists, modify it. If it does not exist, add a new occurrence like WHIS_TYPE_ADDNEW.

recid

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record that contains the history repeat area.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of a field in the history repeat area (which must have a valid occurrence number, such as 1) or repeat area count field tag (which must have an occurrence number of zero).

numfts

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numfts* is the number of fields to be written for each occurrence. *numfts* is the number of array elements in *fts*, *datats*, and *ptdatas*.

fts

Data Type	long word array
Access	Input only
Mechanism	Passed by reference
Description	*fts* is an array of the field tags of the fields to be written. The occurrence number part of each field tag must be 0.

datats

Data Type	short word array
Access	Input only
Mechanism	Passed by reference
Description	*datats* is an array of the data types of the fields to be read.

ptdatas

Data Type	pointer array
Access	Input only
Mechanism	Passed by reference
Description	*ptdatas* is an array of pointers to data, one pointer for each field to be written. Each data (of the type specified by its *datats* element) contains the data to be written to the field in the occurrence.

keylevel

Data Type	short word
Access	Input
Mechanism	Passed by value
Description	*keylevel* holds the key quality level of the occurrence being written.. This parameter can be set to –1, in which case the key level of the occurrence is not set explicitly.

keytime

Data Type	XUSTS
Access	Input
Mechanism	Passed by reference
Description	*keytime* holds the key time for the occurrence to update or create.

ftsok

Data Type:	short word
Access:	Output only
Mechanism:	Passed by reference
Description:	*ftsok* returns the number of fields written. *ftsok* returns *numfts* unless there is an error.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* returns an error code as defined in the **setcim.h** include file.

Sample C Program

```
long        id;
long        fts[2] = { 0x241A0000, 0x24190000 };
short       data_ts[2] = { DTYPXTIM, DTYPDUBL };
XTSBLOCK    times[MAXOCCS];
double      vals[MAXOCCS];
XUSTS               keytime;
short       keylevel;
void        *ptdatas[2] = { (void *)times, (void *)vals };
XUSTS       xusts_old;
XUSTS       xusts_new;
long        seq;
long        occs_ok;
short       fts_ok;
short       stop_code;
ERRBLOCK    err;

WHIS21DAT (WHIS_TYPE_UPDATE, id, fts[0] + 1, 2, fts, data_ts, ptdatas,
      keylevel, &keytime,  &fts_ok,  &err );
```

.

Sample FORT*R*AN Program

```
INTEGER*4 ID
INTEGER*4 FTS(2)
INTEGER*2 DATA_TS(2)
RECORD/XTSBLOCK/ TIMES(MAXOCCS)
RECORD/XUSTS/ KEYTIME
REAL*8 VALS(MAXOCCS)
INTEGER*2 KEYLEVEL
INTEGER*4 PTDATAS(2)
INTEGER*2 FTS_OK
RECORD/ERRBLOCK/ ERR


PTDATAS(1) = %LOC(TIMES(1))
PTDATAS(2) = %LOC(VALS(1))
CALL WHIS21DAT( %VAL(WHIS_TYPE_UPDATE),
       %VAL(ID), %VAL(FTS(1) + 1), %VAL(2), %REF(FTS),
%REF(DATA_TS),
       %REF(PTDATAS), %VAL(KEYLEVEL), KEYTIME, FTS_OK, ERR )
```

XTIM2DB

Writes an extended timestamp to the database (data type = DTYPXTIM).

Format

```
XTIM2DB(recid, ft, xtsdata, error)
```

Arguments

recid

Data Type	long word
Access	Input only Input only Input onl Input only
Mechanism	Passed by value
Description	*recid* is the ID of the record containing the field.

ft

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*ft* is the field tag of the field to write into.

xtsdata

Data Type	XTSBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*xtsdata* is the extended timestamp to write.

error

Data Type	ERRBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*error* is the error code as defined in the **setcim.h** include file.

Sample C Program

```
long        recid,      /* Record ID          */
            ft;         /* Field Tag          */
ERRBLOCK    err;        /* Error return status      */
XTSBLOCK    xtsdata;          /* Extended timestamp     */


XTIM2DB (recid, ft, &xtsdata, &err);
```

Sample FORTRAN Program

```
INTEGER*4   RECID
INTEGER*4   FT
RECORD              /ERRBLOCK/ERR
RECORD              /XTSBLOCK/XTSDATA


CALL XTIM2DB( %VAL(RECID), %VAL(FT), XTSDATA, ERR)
```

XTS2ASCII

Converts an extended timestamp to the current Aspen InfoPlus.21 ASCII date/time format.

Format

```
XTS2ASCII(xts, ptbuff, sizebuff, error)
```

Arguments

xts

Data Type	XTSBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*xts* is the extended timestamp to format.

ptbuff

Data Type	character array
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* specifies the address of the buffer to receive the ASCII form of the timestamp.

sizebuff

Data Type	short word

Access	Input only
Mechanism	Passed by value
Description	*sizebuff* specifies the number of characters in the buffer. Valid sizes are 15, 18, and 20. Any other size causes *error* to be set to 1.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*error* is set to 0 if no error. Otherwise *error* is set to 1.

Sample C Program

```
char        ptbuff[20];    /*  Address of buffer to receive ASCII data*/
                           /*  the ASCII data                          */
short       sizebuff;           /*  Size of the buffer                 */
char        error;         /*  Returns TRUE if timestamp is invalid */
XTSBLOCK    xts;           /*  Extended timestamp                     */

XTS2ASCII (&xts, ptbuff, sizebuff, &error);
```

Sample FORTRAN Program

```
CHARACTER*20      PTBUFF
INTEGER*2   SIZEBUFF
BYTE        ERROR
RECORD            /XTSBLOCK/XTS

CALL XTS2ASCII (XTS, %REF(PTBUFF), %VAL(SIZEBUFF), ERROR)
```

XTS2DSPDT

Converts an Aspen InfoPlus.21 extended timestamp to the day of the century and the time. **XTS2DSPDT** automatically incorporates the Aspen InfoPlus.21 system time offset into the conversion.

Format

```
XTS2DSPDT(xts, dspdate, dsptime)
```

Arguments

xts

Data Type	XTSBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*xts* is the Aspen InfoPlus.21 extended timestamp to be converted.

dspdate

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*dspdate* is the returned number of days into the century.

dsptime

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*dsptime* is the returned number of tenths of seconds into the day of the century.

Sample C Program

```
XTSBLOCK    xts;          /*  Extended timestamp to be converter  */
long        dspdate,            /*  Returned # of days into century*/
            dsptime;            /*  Returned # of 1/10 second into day
      */

XTS2DSPDT (&xts, &dspdate, &dsptime);
```

Sample FORTRAN Program

```
RECORD          /XTSBLOCK/XTS
INTEGER*4   DSPDATE
INTEGER*4   DSPTIME

CALL XTS2DSPDT (XTS, DSPDATE, DSPTIME)
```

5 Utility Subroutines

This chapter gives a brief description of subroutines that do not access the Aspen InfoPlus.21 database but are useful in developing Aspen InfoPlus.21 applications. Each description gives enough information to determine what routines are available and, in general, which should be used for a given application.

Utilities for Dealing with Time and Timestamp

ASCII2DTIM

Converts a signed ASCII delta time to a signed time interval in units of 0.1 second.

CALCDELXTM

Returns the delta time (in .1 sec) between two Aspen InfoPlus.21 extended timestamps as a double-precision real number.

DTIM2ASCII

Converts a signed time interval in units of 0.1 second to a signed ASCII delta time

XTSPLUSDT

Adjusts an Aspen InfoPlus.21 extended timestamp by a delta time, in tenths of seconds.

XTS2XUST

Converts an extended Aspen InfoPlus.21 timestamp to an extended microsecond (or UNIX-style) timestamp.

XUST2XTS

Converts an extended microsecond (or UNIX-style) timestamp to an extended Aspen InfoPlus.21 timestamp.

Utilities for Converting to ASCII

DTIM2ASCII

Converts a signed time interval in units of 0.1 second to a signed ASCII delta time.

DUBL2ASCII

Converts a double-precision real number to an ASCII representation of the number.

Utilities for Converting from ASCII

ASCII2DTIM

Converts a signed ASCII delta time to a signed time interval in units of 0.1 second.

ASCII2DUBL

Converts an ASCII representation of a real number to a double-precision real number.

Utilities for Special Floating Point Numbers

ASCII2DUBL

Converts the ASCII representation of a real number to a double-precision real number.

DSPECGET

Obtains a special double-precision real number.

DSPECTYP

Obtains the type of a double-precision real number.

DUBL2ASCII

Converts a double-precision real number to ASCII.

RSPECGET

Obtains a special single-precision real number.

RSPECTYP

Obtains the type of a single-precision number.

Utility Subroutine Abstracts

The utility routines in this chapter do not access the Aspen InfoPlus.21 database but can be useful in programs that do. These routines:

- convert data to/from ASCII representations
- compare two buffers or character strings
- initialize values in a buffer or a character string
- move data from one buffer to another
- shift data within a buffer
- convert timestamp formats
- calculate a delta time from two timestamps

ASCII2DTIM

Converts a signed ASCII delta time to a signed time interval in units of 0.1 second.

Format

```
ASCII2DTIM(ptbuff, numchars, time, error)
```

Arguments

ptbuff

Data Type	byte
Access	Input only
Mechanism	Passed by reference

Description:	*ptbuff* is the buffer containing the ASCII time in the format ±HHHHH:MM:SS.T. The following values are allowed. The "+" or "–" sign is optional: Field — Value HHHHH — 0 to 17531 MM — 0 to 59 SS — 0 to 59 T — 0 to 9

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of ASCII characters in *ptbuff* to use. If *ptbuff* is signed, *numchars* must be ≥ 6. If *ptbuff* is unsigned *numchars* must be ≥ 5.

time

Data Type	long word
Access	Output only
Mechanism	Passed by reference
Description	*time* is the number of tenths of a second in the time interval.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*error* = 0 if there is no error. Otherwise, *error* = 1.

Sample C Program

```
char  ptbuff[14]; /*  ASCII time interval       */
short numchars;   /*  Number of ASCII characters to use     */
long  time;       /*  Time interval (in .1sec)        */
```

```
char  error;            /*  = 0 if no error                 */
ASCII2DTIM (ptbuff, numchars, &time, &error);
```

Sample FORTRAN Program

```
CHARACTER*14      PTBUFF
INTEGER*2   NUMCHARS
INTEGER*4   TIME
BYTE        ERROR
CALL ASCII2DTIM (%REF(PTBUFF), %VAL(NUMCHARS), TIME, ERROR)
```

ASCII2DUBL

Converts an ASCII representation of a real number to a double-precision real number.

Format

```
ASCII2DUBL(ptbuff, numchars, realnum, error)
```

Arguments

ptbuff

Data Type	character array
Access	Input only
Mechanism	Passed by reference
Description	*ptbuff* is the buffer containing the ASCII number. If all the characters in *ptbuff* are the same, the following special real numbers may be specified: + Positive infinity – Negative infinity ? Undefined If the last character is "+", "–", or "?", then all the preceding characters must also be that same special character (i.e., "+", "–", or "?"). If *ptbuff* does not contain a special number, then one or more blanks can precede the most significant digit or the ’.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of ASCII characters in *ptbuff* to use.

realnum

Data Type	double-precision real
Access	Output only
Mechanism	Passed by reference
Description	*realnum* is the converted real number.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*error* = 0 if there is no error. Otherwise, *error* = 1.

Sample C Program

```
char        ptbuff[8];        /*  Buffer containing ASCII string
      */
short       numchars;   /*  Number of characters in ASCII string
*/
double            realnum;          /*  Real number
      */
char        error;            /*  Error indicator (0 = no error)
      */

ASCII2DUBL (ptbuff, numchars, &realnum, &error);
```

Sample FORTRAN Program

```
CHARACTER*8       PTBUFF
INTEGER*2   NUMCHARS
REAL*8            REALNUM
BYTE        ERROR
                              .
CALL ASCII2DUBL (%REF(PTBUFF), %VAL(NUMCHARS), REALNUM, ERROR)
                              :
                              .
```

CALCDELXTM

Returns the delta time (in .1 sec) between two Aspen InfoPlus.21 extended timestamps as a double–precision real number.

Return

Returns double–precision real.
Returns the delta time (newtime – oldtime) in units of .1 sec.

Format

```
CALCDELXTM(newtime, oldtime)
```

Arguments

newtime

Data Type	XTSBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*newtime* is the first extended timestamp. **CALCDELXTM** assumes that *newtime* represents a valid timestamp. Consequently, *newtime.XTSSOLD* should be between MINLEAPY and (MINLEAPY + 24) inclusive and *newtime.XTSFAST* should be between 0 and (MAXTIME – 1) inclusive.

oldtime

Data Type	XTSBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*oldtime* is the second extended timestamp. CALCDELXTM assumes that *oldtime* represents a valid timestamp. Consequently, *oldtime.XTSSOLD* should be between MINLEAPY and (MINLEAPY + 24) inclusive and *oldtime.XTSFAST* should be between 0 and (MAXTIME – 1) inclusive.

Sample C Program

```
XTSBLOCK    newtime;    /* First timestamp            */
XTSBLOCK    oldtime;    /* Second timestamp     */
double            deltatime   /* Delta time in .1 sec       */

deltatim = CALCDELXTM (&newtime, &oldtime);
```

Sample FORTRAN Program

```
RECORD/XTSBLOCK/ NEWTIME
RECORD/XTSBLOCK/ OLDTIME
REAL*8                    DELTATIM

DELTATIM = CALCDELXTM(NEWTIME, OLDTIME)
```

DSPECGET

Obtains a special double-precision real number.

Format

```
DSPECGET(type, realnum)
```

Arguments

type

Data Type	SPECTYPF
Access	Input only
Mechanism	Passed by value
Description	*type* is the special real number *type* required.

Symbol	Value	Meaning
NAN	0	Not a number (Undefined)
NEGINFIN	1	Negative Infinity
POSINFIN	2	Positive Infinity
NORMAL	3	Normal (Not a special real number)

r**ealnum**

Data Type	double-precision real
Access	Output only
Mechanism	Passed by reference
Description	*realnum* is the returned double-precision real number.

Sample C Program

```
double      realnum;          /* Double-precision float */
DSPECGET (NAN, &realnum);
```

Sample FORTRAN Program

```
REAL*8            REALNUM
CALL DSPECGET (%VAL(NAN), REALNUM)
```

DSPECTYP

Obtains the type of a double-precision real number.

Returns

SPECTYPF

The value returned indicates the type of real number.

Symbol	Value	Meaning
NAN	0	Not a number (Undefined)
NEGINFIN	1	Negative Infinity
POSINFIN	2	Positive Infinity
NORMAL	3	Normal (Not a special real number)

Format

```
DSPECTYP(realnum)
```

Arguments

realnum

Data Type	double-precision real
Access	Input only
Mechanism	Passed by reference
Description	*realnum* is the double-precision real number to be tested.

Sample C Program

```
double            realnum;          /*  Double-precision float */

if(DSPECTYP(&realnum) == NAN)
{…}
```

Sample FORTRAN Program

```
REAL*8            REALNUM
IF( DSPECTYP(REALNUM) .EQ. NAN ) THEN ...
```

DTIM2ASCII

Converts a signed time interval in units of 0.1 second to a signed ASCII delta time.

Format

```
DTIM2ASCII(time, code, numchars, ptbuff, error)
```

Arguments

time

Data Type	long word
Access	Input only
Mechanism	Passed by value
Description	*time* is the number of tenths of a second in the time interval.

code

Data Type	short word

Access	Input only
Mechanism	Passed by value
Description	*code* specifies the ASCII format to use. Each value for *code* has a corresponding minimum value for *numchars*. Possible combinations are: code — ASCII Format — Min. numchars 0 — HHHHH:MM:SS.T — 10 1 — HHHHH:MM:SS — 8 -1 — HHHHH:MM — 5 Any other combination is considered an error.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of ASCII characters in *ptbuff* to use.

ptbuff

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* is the buffer to contain the ASCII time. The first character in the ASCII time interval is "+" or "−" unless *numchars* is not large enough. In that case, the entire buffer will be filled with ">" (positive *time*) or "<" (negative *time*) characters.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*error* = 0 if there is no error. Otherwise *error* = 1.

Sample C Program

```
long  time;       /*  Interval (in .1sec) to be converted to
ASCII */
short code;       /*  Format specifier                      */
short numchars;   /*  Number of ASCII characters to use
      */
char  ptbuff[14]; /*  ASCII time interval             */
char  error;        /*  = 0 if no error
      */

DTIM2ASCII (time, code, numchars, ptbuff, &error);
```

Sample FORTRAN Program

```
INTEGER*4          TIME
INTEGER*2          CODE
INTEGER*2          NUMCHARS
CHARACTER*14             PTBUFF
BYTE               ERROR

CALL DTIM2ASCII (%VAL(TIME),%VAL(CODE), %VAL(NUMCHARS),
        %REF(PTBUFF),ERROR)
```

DUBL2ASCII

Converts a double-precision real number to an ASCII representation of the number.

Format

```
DUBL2ASCII(realnum, numchars, numwhole, ptbuff, error)
```

Arguments

realnum

Data Type	double-precision real
Access	Input only
Mechanism	Passed by reference
Description	*realnum* is the real number to be converted to ASCII.

numchars

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numchars* specifies the number of ASCII characters in *ptbuff* to use. If *numchars* is not large enough to hold the ASCII equivalent of *realnum*, then *ptbuff* is filled with "<" characters if *realnum* is negative or ">" characters if *realnum* is positive. *numchars* must equal or exceed (*numwhole* + 2) or an error is generated.

numwhole

Data Type	short word
Access	Input only
Mechanism	Passed by value
Description	*numwhole* is the number of whole digits. Assuming *numchars* is large enough, the number of whole digits in the result equals or exceeds that specified by *numwhole*. For example, if *numwhole* is 3 and *realnum* is 2456.7, then 4 whole digits are placed to the left of the decimal point, not 3. A negative value for *numwhole* is considered an error.

ptbuff

Data Type	byte
Access	Output only
Mechanism	Passed by reference
Description	*ptbuff* is the buffer to contain the ASCII number. If *realnum* is undefined, *ptbuff* is filled with "?" characters. If *realnum* is positive infinity, *ptbuff* is filled with "+" characters. If *realnum* is negative infinity, *ptbuff* is filled with "–" characters.

error

Data Type	byte
Access	Output only
Mechanism	Passed by reference

Description	*error* = 0 if there is no error. Otherwise *error* = 1.

Sample C Program

```
double realnum;              /*  Number to be converted to ASCII
        */
short   numchars;    /*  Maximum number of ASCII characters */
short   numwhole;    /*  Number of whole digits             */
char    ptbuff[8];           /*  Buffer containing ASCII string          */
char    error;       /*  Error indicator (0 = no error)     */

DUBL2ASCII (&realnum, numchars, numwhole, ptbuff, &error);
```

Sample FORTRAN Program

```
REAL*8              REALNUM
INTEGER*2    NUMCHARS
INTEGER*2    NUMWHOLE
CHARACTER*8         PTBUFF
BYTE         ERROR

CALL DUBL2ASCII (REALNUM,%VAL(NUMCHARS),%VAL(NUMWHOLE),
        %REF(PTBUFF), ERROR)
```

RSPECGET

Obtains a special single-precision real number.

Format

```
RSPECGET(type, realnum)
```

Arguments

type

Data Type	SPECTYPF
Access	Input only
Mechanism	Passed by value
Description	*type* is the special real number type required.

Symbol	Value	Meaning

Symbol	Value	Meaning
NAN	0	Not a number (Undefined)
NEGINFIN	1	Negative Infinity
POSINFIN	2	Positive Infinity
NORMAL	3	Normal (Not a special real number)

realnum

Data Type	single-precision real
Access	Output only
Mechanism	Passed by reference
Description	*realnum* is the returned single-precision real number.

Sample C Program

```
float        realnum;          /* Single-precision float */

RSPECGET (NAN, &realnum);
```

Sample FORTRAN Program

```
REAL*4            REALNUM

CALL RSPECGET ( %VAL(NAN), REALNUM)
```

RSPECTYP

Obtains the type of a single-precision real number.

Format

```
RSPECTYP(realnum)
```

Returns

SPECTYPF

Symbol	Value	Meaning
NAN	0	Not a number (Undefined)
NEGINFIN	1	Negative Infinity
POSINFIN	2	Positive Infinity
NORMAL	3	Normal (Not a special real number)

Arguments

realnum

Data Type	single-precision real
Access	Input only
Mechanism	Passed by reference
Description	*realnum* is the single-precision real number to be tested.

Sample C Program

```
float          realnum;          /* Single-precision float */

if(RSPECTYP(&realnum) == NAN)
{…}
```

Sample FORTRAN Program

```
REAL*4             REALNUM

IF( RSPECTYP(REALNUM) .EQ. NAN ) THEN ...
```

XTSPLUSDT

Adjusts an Aspen InfoPlus.21 extended timestamp by a delta time, in tenths of seconds.

Format

```
XTSPLUSDT(xtime, deltatime)
```

Arguments

xtime

Data Type	XTSBLOCK
Access	Input/Output
Mechanism	Passed by reference
Description	*xtime* is the extended timestamp to be adjusted by *deltatime*.

deltatime

Data Type	Double-precision Real
Access	Input only
Mechanism	Passed by reference
Description	*deltatime* is the positive or negative number of tenths of seconds.

Sample C Program

```
XTSBLOCK    xtime;            /*  Time to be adjusted       */
double            deltatime;        /*  Tenths of seconds   */

XTSPLUSDT (&xtime, &deltatime);
```

Sample FORTRAN Program

```
RECORD/XTSBLOCK/  XTIME
REAL*8                  DELTATIME
```

```
CALL XTSPLUSDT (XTIME, DELTATIME)
```

XTS2XUST

Converts an extended Aspen InfoPlus.21 timestamp to an extended microsecond (or UNIX-style) timestamp.

Format

```
XTS2XUST(xts, xusts)
```

Arguments

xts

Data Type	XTSBLOCK
Access	Input only
Mechanism	Passed by reference
Description	*xts* is the extended Aspen InfoPlus.21 timestamp to convert from.

xusts

Data Type	XUSTS
Access	Output only
Mechanism	Passed by reference
Description	*xusts* is the extended microsecond timestamp to convert to.

Sample C Program

```
XTSBLOCK     xts;
XUSTS                xusts;

XTS2XUST (&xts, &xusts);
```

Sample FORTRAN Program

```
RECORD/XTSBLOCK/xts
RECORD/XUSTS/xusts
```

XTS2XUST (xts, xusts)

XUST2XTS

Converts an extended microsecond (or UNIX-style) timestamp to an extended Aspen InfoPlus.21 timestamp.

Format

```
XUST2XTS(xusts, xts)
```

Arguments

xusts

Data Type	XUSTS
Access	Input only
Mechanism	Passed by reference
Description	*xusts* is the extended microsecond timestamp from which to convert.

xts

Data Type	XTSBLOCK
Access	Output only
Mechanism	Passed by reference
Description	*xts* is the extended Aspen InfoPlus.21 timestamp to which to convert.

Sample C Program

```
XTSBLOCK     xts;
XUSTS                xusts;

XTS2XUST (&xts, &xusts);
```

Sample FORTRAN Program

```
RECORD/XTSBLOCK/xts
RECORD/XUSTS/xusts;


XUST2XTS (xusts , xts)
```

6 Remote Access Functions

Server functions are called by programs that need to control which remote API servers are accessed. Programs that call these functions should include the header file, **infoplus21_api.h**. These functions are available in the **infoplus21_api.dll** library.

Default Node Routines

Not all database access routines have a record ID argument in which to specify a node. Therefore, database functions such as **GETDBXTIM** use the default node. Each database node listed in the configuration file is numbered. The first node listed is set as the default node for the client program. In order to change the default node at runtime, use the following functions:

int SET_DEFAULT_NODE – Changes the default node given a node number

int SET_DEFAULT_ALIAS – Changes the default node given an alias name

int GET_DEFAULT_NODE – Returns the default node number

int GET_DEFAULT_ALIAS – Returns the alias name of the default node

Dynamically Selecting Data Sources

When a client program uses the Aspen InfoPlus.21 API Server, the infoplus21_api.dll library provides several special functions:

DaSelectServer

DaSelectServer(TdaServerIndex ServerIndex, ERRBLOCK* ErrorBlock)

If you have more than one server in a file, and are trying to resolve record names and id names, the API will search all that are in the file. To search on a specific database only, use **DaSelectServer**.

API calls such as DECODNAM or NAME2RECID search all connected databases for a given record name. If they find a database containing the specified

record they will return the long record ID. Part of the 16 most significant bits contain the node ID and the 16 least significant bits contain the ID of the record within that database.

You can use the function DaSelectServer, if:

- the client program wants to query a specific database for a record name.
- the client program wants record ID returned if the specified record name is in the database and an error if it is not.

This function will limit all calls to the server specified by the ServerIndex parameter. As a result, the specified server will become the default, and current server for the calling client. If the client wants to switch to another server, it can call **DaSelectServer** again with the corresponding ServerIndex parameter.

DaAddServer

DaServerIndex **DaAddServer**(const char* ConfigLine, ERRBLOCK* ErrorBlock)

As documented in this manual, at startup, a client program can read information from the configuration file SETCIMRPC.CFG to make connections to all the listed databases. The function **DaAddServer** will allow client programs to:

- connect to additional databases that are not listed in the configuration file initially.
- connect to databases only when needed and not use the configuration file at all.

The ConfigLine parameter has the same syntax as a line in the configuration file. If successful, **DaAddServer** will return the server index of the database.

DaSetDefaultServer

TdaBoolean **DaSetDefaultServer**(TdaServerIndex ServerIndex, ERRORBLOCK* ErrorBlock)
This function sets the specified server to be the default server for the calling thread. The default server is used for calls that do not contain node information.

GET_DEFAULT_NODE()

long **GET_DEFAULT_NODE()**
This function returns the server index of the default server for the calling thread.

SET_DEFAULT_NODE

int **SET_DEFAULT_NODE**(TdaServerIndex ServerIndex, ERRBLOCK* ErrorBlock)
This function is another variation of DaSetDefaultServer.

Macros Defined in INFOPLUS21_API.H

long SETNODEID – Embeds a node ID into a record ID

long GETRECID – Strips a node ID from a record ID

long GETNODEID – Returns the node ID embedded in a record ID

7 FORTRAN Access to the Aspen InfoPlus.21 API

The sections in this chapter discuss how the Aspen InfoPlus.21 API can be accessed from modules written in FORTRAN. The compiler used to implement and test the FORTRAN interface was a Digital Visual FORTRAN V5.0B; other compilers may work differently. Refer to the compiler documentation explaining how to implement calls to other languages if you are not using the Digital Visual FORTRAN V5.0B.

The Aspen InfoPlus.21 API has been implemented in the C programming language. Because C and FORTRAN follow different rules when implementing a subroutine or function call, some special attention is needed when calling Aspen InfoPlus.21 API routines from FORTRAN programs.

Digital Visual FORTRAN has provided the ability to "prototype" subroutine calls to help make this process easier. This feature has long been available in the C language and provides a way to define an external subroutine and how it is to be called. With this definition available, the compiler can check all procedure calls to make sure the call has the proper number and type of arguments. With Digital Visual FORTRAN, the prototype syntax allows you to define which calling arguments are being passed by reference (as a pointer) and which are being passed by value to the C function. When writing the actual call in a program, there is no need to be concerned with which arguments are passed by reference and which are passed by value, the compiler knows from the prototype and takes care of it for you. Prototypes for most of the Aspen InfoPlus.21 API routines are provided in the **SETCIM.INC** include file.

FORTRAN vs. C

The FORTRAN and C programming languages have taken almost opposite approaches to how arguments are passed between modules. FORTRAN passes all subroutine and function arguments by reference, meaning that FORTRAN passes the memory location of an argument in the form of a pointer to a subroutine. The FORTRAN subroutine can then obtain and/or change the value of the original variable by referencing this location. Different FORTRAN compilers also handle passing structures and CHARACTER variables in

different ways. FORTRAN on DEC VMS systems pass "descriptors" for these special types of data.

Most FORTRAN compilers provide some way to overcome the default "pass by reference." The DEC VMS FORTRAN compiler uses the %VAL, %REF, and %DESC directives to tell the compiler to pass specific arguments as values, pointers, or descriptors.

The C language defaults to passing singular arguments by value, meaning that (in most cases) the value of an argument is passed to a subroutine. The subroutine cannot change this value because it does not know the location of the original argument. In order to allow a subroutine to change the value of an argument, the programmer has to explicitly pass a pointer to the argument. A C function may have a mix of arguments that are passed by value and by reference. In general, only those arguments that will be changed by the function will be passed by reference. Arrays and structures are automatically passed by reference in C because of their special needs. Refer to a C programming guide for more information.

The other major difference between C and FORTRAN is that C only has "functions." In C, a typed function is similar to a FORTRAN function and is used the same way. An untyped C function is similar to a FORTRAN subroutine and is called from FORTRAN like a subroutine.

Function and Subroutine Prototypes

Details on this topic can be found in the Digital Visual FORTRAN online documentation.

The INTERFACE block is used to specify function and subroutine prototypes in Visual FORTRAN. The following example is copied from the **SETCIM.INC** file and shows the prototype for the **DB2LONG** function:

```
INTERFACE

        SUBROUTINE DB2LONG (RECID, FT, INTDATA, ERROR)
          !DEC$ATTRIBUTES C, ALIAS: '_DB2LONG' :: DB2LONG
          !DEC$ATTRIBUTES REFERENCE :: INTDATA
          !DEC$ATTRIBUTES REFERENCE :: ERROR
          !DEC$ATTRIBUTES VALUE :: RECID,FT
          INTEGER*4 RECID
          INTEGER*4 FT
          INTEGER*4 INTDATA
          structure    /ERRBLOCK/
           integer*2 ERRCODE
           integer*4 ERR1
           integer*4 ERR2
          end structure
          RECORD    /ERRBLOCK/  ERROR
        END SUBROUTINE

END INTERFACE
```

This block of code defines how the subroutine **DB2LONG** is to be called. More specifically, there are four arguments, the first three are four-byte INTEGERs and the last is defined by the ERRBLOCK structure. If **DB2LONG** was a FORTRAN module, the compiler would have all the information it needed to know. But, **DB2LONG** is implemented in C. The lines beginning with "!DEC$ATTRIBUTES" help the FORTRAN compiler make the call compatible with the C calling standards.

The first !DEC$ATTRIBUTES line tells the FORTRAN compiler that **DB2LONG** is a C function and that when compiling the call use the symbol "_DB2LONG" to reference the function. The next two lines tell the compiler that the INTDATA and ERROR arguments are passed by reference. The next line tells the compiler that the RECID and FT arguments are passed by value.

Calling a C Function

The code sample below shows how the **DB2LONG** routine would be called in a FORTRAN program. It is assumed that valid values are placed in RECID and FT before the call is made.

```
      INTEGER*4  RECORD, FIELDT, VALUE
      RECORD    /ERRBLOCK/  ERROR
     .
     .
     .
      CALL DB2LONG (RECORD, FIELDT, VALUE, ERROR)
```

Nothing special has been done in the actual code because the prototype has defined how the call is to be made for the compiler. Instead of passing RECORD and FIELDT by reference, the compiler will generate code that will pass them by value.

This approach differs from past implementations of FORTRAN where special directives were needed in the call to insure the arguments were passed properly. For instance, with DEC VMS FORTRAN, the call would be as follows:

```
CALL DB2LONG (%VAL(RECORD), %VAL(FIELDT), VALUE, %REF(ERROR))
```

This syntax will still work. If you have existing code that uses it, there is no need to change it. But if you are writing new code, it is not necessary to use this syntax (unless you want to insure compatibility with other older machines).

Index

F

G

H

I

L

M

U

V

W

X