**International Crop Information System**

10

**TDM Windows Application Programming Interface 5.4**

From ICISWiki

**McLaren, Graham (GCP)**

Contents

[**DLL Architecture** 3](#_Toc413928372)

[**INTRODUCTION** 3](#_Toc413928373)

[**ICIS Data Structures** 4](#_Toc413928374)

[**Introduction** 4](#_Toc413928375)

[**GMS DATA STRUCTURES** 5](#_Toc413928376)

[**LIST DATA STRUCTURE** 8](#_Toc413928377)

[**LMS Data Structure** 9](#_Toc413928378)

[**DMS DATA STRUCTURE** 10](#_Toc413928379)

[**IMS DATA STRUCTURES** 14](#_Toc413928380)

[**BIBREF DATA STRUCTURE** 15](#_Toc413928381)

[**GEMS DATA STRUCTURE** 16](#_Toc413928382)

[**SUPPORTING TABLES** 20](#_Toc413928383)

[**DLL Functions for Access Control** 21](#_Toc413928384)

[**GENEALOGY MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS** 22](#_Toc413928385)

[**DATA MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS** 24](#_Toc413928386)

[**INVENTORY MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS** 25](#_Toc413928387)

[**GENE MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS** 26](#_Toc413928388)

[**USER AND INSTALLATION** 27](#_Toc413928389)

[**GMS DLL FUNCTIONS** 28](#_Toc413928390)

[**SEARCH FUNCTIONS** 28](#_Toc413928391)

[**GMS DATA RETRIEVAL functions** 33](#_Toc413928392)

[**Functions to add data to the local database** 41](#_Toc413928393)

[**Functions to correct LOCAL GMS records** 44](#_Toc413928394)

[**BASIC GMS FUNCTIONS** 48](#_Toc413928395)

[**LIST DLL FUNCTIONS** 54](#_Toc413928396)

[**FUNCTIONS ASSOCIATED TO LISTNMS TABLE** 54](#_Toc413928397)

[**FUNCTIONS ASSOCIATED WITH LISTDATA TABLE** 57](#_Toc413928398)

[**USER DEFINED FIELDS AND CONSTANTS FUNCTIONS** 59](#_Toc413928399)

[**DMS DLL FUNCTIONS** 61](#_Toc413928400)

[**INTRODUCTION** 61](#_Toc413928401)

[**ADD FUNCTIONS** 62](#_Toc413928402)

[**DMS FIND FUNCTIONS** 68](#_Toc413928403)

[**DMS GENERAL QUERY FUNCTIONS** 73](#_Toc413928404)

[**QUERY THROUGH OBSERVATION UNIT** 75](#_Toc413928405)

[**DMS ACCESS FUNCTIONS** 82](#_Toc413928406)

[**DMS UPDATE FUNCTIONS** 96](#_Toc413928407)

[**DMS DELETE FUNCTION** 97](#_Toc413928408)

[**MISCELLANEOUS FUNCTIONS** 98](#_Toc413928409)

[**IMS DLL FUNCTIONS** 101](#_Toc413928410)

[**ADD Functions** 101](#_Toc413928411)

[**DATA RETRIEVAL FUNCTIONS** 102](#_Toc413928412)

[**UPDATE FUNCTIONS** 106](#_Toc413928413)

[**REPORT FUNCTIONS** 107](#_Toc413928414)

[**MISCELLANEOUS** 113](#_Toc413928415)

[**LMS DLL FUNCTIONS** 114](#_Toc413928416)

[**BIBREF DLL FUNCTIONS** 117](#_Toc413928417)

[**GEMS DLL FUNCTIONS** 118](#_Toc413928418)

[**ADD FUNCTIONS** 118](#_Toc413928419)

[**GET FUNCTIONS** 125](#_Toc413928420)

**DLL Architecture**

**INTRODUCTION**

ICIS applications can be written in any computer language under a WINDOWS 32-bit operating system. They must only access the ICIS databases through the ICIS DLL functions. Programs, which access the database directly or through any other system, may be useful for loading data, but are not portable to other users and so, are not part of the ICIS system. Any applications, which have potential value for other users, should only use the DLL.

**Conventions of ICIS Application Development**

Most ICIS applications are expected to be WINDOWS 95, 32-bit programs but they could be WINDOWS NT, Windows 2000 or 32-bit console applications. The restriction to 32 bit operating systems is due to requirements of the ICIS DLL and the ODBC drivers it uses. This is not regarded as a serious restriction since ICIS will at least require a computer with a CD drive, and most of these are capable of running a 32-bit operating system. Apart from this restriction applications can be written in any computer language which can access the ICIS DLL. FORTRAN 90, Delphi PASCAL, Visual Basic, and C++ have been used to access the DLL so far. The first function of any ICIS application is to establish the ODBC data connections for the current session. To do this it must read the ODBC data source names from an INI file. The default file is called ICIS.INI. An application initially looks for ICIS.INI in the WINDOWS %TEMP% directory (the name of this directory is stored in the environment variable temp), unless a parameter is passed together when an application is called. The parameter to use a different INI file should contain the full pathname of the INI file to use. The structure of ICIS.INI files is defined in 5.3.1, but new applications may require new sections in the file. These sections should be headed by the application name and will be ignored by other applications. Since several users and projects may wish to access different data sources on the same computer a system of replacing ICIS.INI for each session has been established through the LAUNCHER application (5.3). The second function of any application is to identify the user and verify his or her access privileges.

**Data Source Names and the INI file**

Access to ICIS databases is provided through the ICIS DLL described in this chapter. Functions in this DLL use ODBC to access the data in the Central and Local databases. These must therefore be defined as ODBC data sources before ICIS applications will operate. Steps to set up the sources are described in 5.2.3b. The ICIS DLL obtains information from an initialization file. This file must exist before applications will run, the application passes the name of the INI file to the DLL, but for general applications this name should be ICIS.INI.

**Return values for ICIS DLL functions**

Most DLL functions return a long integer value which are: GMS\_SUCCESS if the function call was successful, GMS\_ERROR if the function call results in an error, GMS\_NO\_DATA if no data is found in the databases relevant to a particular call or GMS\_NOACCESS if the ICIS user does not have access to the particular function being called. Some functions return other values to indicate different events. These are described in the function descriptions. Note: When making DLL function calls, always initialize character arguments to null prior to calling the function. If this is not done each and every time, errors may occur or you may accidentally filter your results.

GMS FUNCTIONS return value

|  |  |
| --- | --- |
| **Return Code** | **Value** |
| GMS\_NO\_DATA | 0 |
| GMS\_SUCCESS | 1 |
| GMS\_ERROR | -1 |
| GMS\_NO\_ACCESS | -2 |
| GMS\_INVALID\_PASSWORD | -3 |
| GMS\_INVALID\_USER | -4 |
| GMS\_NAME\_EXISTS | -5 |
| GMS\_NOT\_FOUND | -6 |
| GMS\_MORE\_DATA | -7 |
| GMS\_READ\_ONLY | -8 |
| GMS\_INVALID\_INSTALLATION | -9 |
| GMS\_INVALID\_DBSTRUCTURE | -10 |

**ICIS Data Structures**

**Introduction**

Several compound structures are used to pass information between the DLL and the application. Users may specify their own names for elements in the structures, but not their order data types. However we recommend using the field names supplied to ensure uniformity across applications.

The structures are defined in the following tables. The data type “long” is a 4-byte signed integer, the type “double” is an 8-byte real and the type “single” is a 4-byte real or ‘float’ data type. Character or string elements are denoted char [n] where n is the length of the element in bytes. All character variables are NULL terminated strings so one byte will be used for the NULL character.

The size of elements in the structures is larger than the space allocated to corresponding fields in the database. This facilitates data transfer to the DLL by accommodating the NULL termination character for character variables and by ensuring data alignment in most programming languages. However, care must be taken that data passed to the database through the structures is not too large for the database fields.

ICIS DATA TYPES

|  |  |
| --- | --- |
| **Data Type** | **Size in bytes** |
| Double | 8 |
| single or float | 4 |
| Long | 4 |
| char[n] (including null character) | n |

**GMS DATA STRUCTURES**

**GMS\_Germplasm structure (in order of declaration)**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| GERMPLASM\_ID | long | GID |
| METHOD | long | METHNO |
| NO\_PROGENITORS | long | GNPGS |
| PROGENITOR\_ID1 | long | GPID1 |
| PROGENITOR\_ID2 | long | GPID2 |
| USER | long | GERMUID |
| LOCAL\_GID | long | LGID |
| LOCATION | long | LOCNO |
| DATE | long | GDATE |
| REFERENCE | long | GREF |
| REPLACE | long | GRPLCE |
| MANAGEMENT\_GID | long | MGID |

**GMS\_NameData structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| NAME\_IDENTIFIER | long | NNID |
| GERMPLASM\_ID | long | GID |
| NAME\_TYPE | long | NTYPE |
| STATUS | long | NSTATUS |
| USER | long | NUID |
| LOCATION | long | NLOCNO |
| DATE | long | NDATE |
| REFERENCE | long | NREF |

**GMS\_Attribute structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| AID | long | Attribute identifier |
| GERMPLASM | long | GERMPLASM\_ID |
| TYPE | long | ATTRIBUTE\_TYPE |
| USER | long | ATTRIBUTE\_USER\_ID |
| LOCATION | long | ATTRIBUTE\_LOCATION\_NO |
| DATE | long | ATTRIBUTE\_DATE |
| REFERENCE | long | NAME\_REFERENCE |

**GMS\_Method structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| METHOD\_ID | long | METHOD\_ID |
| REFERENCE | long | METHOD\_REFERENCE |
| NO\_PROGENITORS | long | NO\_PROG\_REQUIRED |
| NO\_MATERNAL | long | NO\_FEMALE\_PARENTS |
| ATTRIBUTE | long | METHOD\_ATTRIBUTE |
| TYPE | char[4] | METHOD\_TYPE |
| CODE | char[12] | METHOD\_CODE |
| NAME | char[52] | METHOD\_NAME |
| GROUP | char[4] | METHOD\_GROUP |
| GENEQ | long | GENETIC\_EQUIVALENCE |
| MUID | long | METHOD\_USER |
| LMID | long | LOCAL\_METHOD\_ID |
| MDATE | long | METHOD\_DATE |

**GMS\_UDField structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| UDFIELD\_ID | long | FLDNO |
| TABLE | char[28] | FTABLE |
| TYPE | char[16] | FTYPE |
| CODE | char[12] | FCODE |
| NAME | char[52] | FNAME |
| FORMAT | char[256] | FFMT |
| LFLDNO | long | LFLDNO |
| FUID | long | FUID |
| FDATE | long | FDATE |
| SCALEID | long | SCALEID |

**GMS\_Changes Structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| CID | long | CID |
| CRECORD | long | CRECORD |
| CFROM | long | CFROM |
| CTO | long | CTO |
| CDATE | long | CDATE |
| CTIME | long | CTIME |
| CUID | long | CUID |
| CREF | long | CREF |
| CTABLE | char[16] | CTABLE |
| CFIELD | char[16] | CFIELD |
| CGROUP | char[20] | CGROUP |

**GMS\_User structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| USER\_ID | long | USER\_ID |
| INSTALLATION | long | INSTALLATION |
| STATUS | integer | USER\_STATUS |
| ACCESS | long | USER\_ACCESS |
| TYPE | char[32] | USER\_TYPE |
| NAME | char[32] | USER\_NAME |
| PASSWORD | char[12] | USER\_PASSWORD |
| PERSON | long | PERSON\_ID |
| ASSIGN\_DATE | long | ASSIGN\_DATE |
| CLOSE\_DATE | long | CLOSE\_DATE |

**GMS\_Installation Structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| INSTALLATION | long | INSTALLATION\_NO |
| ADMINISTRATOR | long | ADMIN\_USER\_ID |
| UDATE | long | UPDATE\_DATE |
| UGID | long | UPDATE\_GID |
| ULOCN | long | UPDATE\_LOCN |
| UMETHN | long | UPDATE\_METHN |
| UFLDNO | long | UPDATE\_FLDNO |
| UREFNO | long | UPDATE\_REFNO |
| UPID | long | UPDATE\_PID |

**LIST DATA STRUCTURE**

**GMS\_ListName Structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| LISTID | long | LISTID |
| LISTNAME | char[48] | LISTNAME |
| LISTDATE | long | LISTDATE |
| LISTTYPE | char[8] | LISTTYPE |
| LISTUID | long | LISTUID |
| LISTDESC | char[80] | LISTDESC |
| LISTSTATUS | long | LIST\_STATUS |
| LISTHIER | long | LIST\_HIERARCHY |

**GMS\_ListData Structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| LISTID | Long | LISTID |
| GID | Long | GID |
| LRECID | Long | LRECID |
| ENTRYCD | char[48] | ENTRYCD |
| SOURCE | char[48] | SOURCE |
| DESIG | char[256] | DESIG |
| GRPNAME | char[256] | GRPNAME |
| ENTRYID | Long | ENTRYID |

**LMS Data Structure**

**GMS\_Location structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name (Table 4.3.1)** |
| LOCATION\_ID | long | LOCATION\_ID |
| TYPE | long | LTYPE |
| NLLP | long | NLLP |
| PREFERRED\_NAME | char[64] | PREFERRED\_NAME |
| ABBREVIATION | char[12] | ABBREVIATION |
| SUBNAT\_LEVEL3 | long | SUBNAT\_LEVEL3 |
| SUBNAT\_LEVEL2 | long | SUBNAT\_LEVEL2 |
| SUBNAT\_LEVEL1 | long | SUBNAT\_LEVEL1 |
| COUNTRY | long | COUNTRY\_CODE |
| REPLACE | long | LOCATION\_REPLACE |

**DMS DATA STRUCTURE**

**DMS\_STUDY Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| STUDYID | Long | STUDYID |
| STUDY\_NAME | Char[52] | SNAME |
| PROGRAM\_KEY | Long | PMKEY |
| TITLE | Char[256] | TITLE |
| INVESTIGATOR\_ID | Long | INVESTID |
| STUDY\_TYPE | Char[2] | STYPE |
| START\_DATE | Long | SDATE |
| END\_DATE | Long | EDATE |
| USERID | Long | USERID |

**DMS\_FACTOR Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| LABELID | Long | LABELID |
| FACTORID | Long | FACTORID |
| FACTOR\_NAME | Char[52] | FNAME |
| STUDYID | Long | STUDYID |
| TRAITID | Long | TRAITID |
| SCALEID | Long | SCALEID |
| METHOD\_ID | Long | TMETHID |
| LEVEL\_TYPE | Char[2] | LTYPE |

**DMS\_VARIATE Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| VARIATID | Long | VARIATID |
| VARIATE\_NAME | Char[52] | VNAME |
| VARIATE\_TYPE | Char[4] | VTYPE |
| STUDYID | Long | STUDYID |
| TRAITID | Long | TRAITID |
| SCALEID | Long | SCALEID |
| METHOD\_ID | Long | TMETHID |
| DATA\_TYPE | Char[2] | DTYPE |

**DMS\_DATAN Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| OBSERVATION\_UNIT | Long | OUNITID |
| VARIATID | Long | VARIATID |
| DATA\_VALUE | Double | DVALUE |

**DMS\_DATAC Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| OBSERVATION\_UNIT | Long | OUNITID |
| VARIATID | Long | VARIATID |
| DATA\_VALUE | Char[256] | DVALUE |

**DMS\_LEVELC Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| LABELID | Long | LABELID |
| FACTORID | Long | FACTORID |
| LEVELNO | Long | LEVELNO |
| LEVEL\_VALUE | Char[256] | LVALUE |

**DMS\_LEVELN Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| LABELID | Long | LABELID |
| FACTORID | Long | FACTORID |
| LEVELNO | Long | LEVELNO |
| LEVEL\_VALUE | Double | LVALUE |

**EFFECT Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| REPRESENTATION\_NO | Long | REPRESNO |
| FACTORID | Long | FACTORID |
| EFFECTID | Long | EFFECTID |

**DMS\_VEFFECT**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| REPRESENTATION\_NO | Long | REPRESNO |
| VARIATID | Long | VARIATID |

**DMS\_OINDEX Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| OUNITID | Long | OUNITID |
| FACTORID | Long | FACTORID |
| LEVELNO | Long | LEVELNO |

**DMS\_TRAIT Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| TID | Long | TID |
| TRAITID | Long | TRAITID |
| TRAIT\_NAME | Char[52] | TRNAME |
| TRAIT\_ABBR | Char[8] | TRABBR |
| TR\_NAME\_STAT | Long | TNSTAT |
| SCALEID | Long | SCALEID |
| METHOD\_ID | Long | TMETHID |

**DMS\_SCALE Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| SCALEID | Long | SCALEID |
| TRAITID | Long | TRAITID |
| SCALE\_NAME | Char[32] | SCNAME |
| SCALE\_TYPE | Char[2] | SCTYPE |

**DMS\_TMETHOD Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| METHOD\_ID | Long | TMETHID |
| TRAITID | Long | TRAITID |
| METHOD\_NAME | Char[52] | TMNAME |
| METHOD\_ABBR | Char[8] | TMABBR |

**DMS\_SCALECON**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| SCALEID | Long | SCALEID |
| SLEVEL | Long | SLEVEL |
| ELEVEL | Long | ELEVEL |

**DMS\_SCALEDIS**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| SCALEID | Long | SCALEI |
| VALUE | Char[24] | VALUE |
| VALDESC | Char[256] | VALDESC |

**DMS\_NumVALUE Structure**

|  |  |
| --- | --- |
| **Element Name** | **Type** |
| ounitid | Long |
| traitid | Long |
| scaleid | Long |
| tmethid | Long |
| nvalue | Double |

**DMS\_CharVALUE Structure**

|  |  |
| --- | --- |
| **Element Name** | **Type** |
| ounitid | Long |
| traitid | Long |
| scaleid | Long |
| tmethid | Long |
| cvalue | Char[256] |

**Source**

|  |  |
| --- | --- |
| **Element Name** | **Type** |
| factorid | Long |
| levelno | Long |

**DMS\_DMSATTR**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| ATTRIBUTE\_ID | Long | DMSATID |
| ATTRIBUTE\_TYPE | Long | DMSATYPE |
| TABLE\_NAME | Long | DMSATAB |
| RECORDID\_IN\_TABLE | Long | DMSATREC |

**IMS DATA STRUCTURES**

**IMS\_LOT Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| LOTID | Long | LOTID |
| LOTUSER | long | USERID |
| ETYPE | Char[16] | ETYPE |
| EID | Long | EID |
| LOCID | Long | LOCID |
| SCALE | Long | SCALEID |
| STATUS | Long | STATUS |
| SOURCEID | Long | SOURCEID |
| COMMENTS | Char[256] | COMMENTS |

**IMS\_TRANSACTION Structure**

|  |  |  |
| --- | --- | --- |
| **Element Name** | **Type** | **Database Field Name** |
| TRNID | Long | TRNID |
| TRNUSER | Long | USERID |
| LOTID | Long | LOTID |
| TRNDATE | Long | TRNDATE |
| TRNSTAT | Long | TRNSTAT |
| CMTDATE | Long | CMTDATE |
| COMMENTS | Char[256] | COMMENTS |
| TRNQTY | Double | TRNQTY |
| PREVAMOUNT | Double | PREVAMOUNT |
| SOURCETYPE | Char[12] | SOURCETYPE |
| SOURCEID | Long | SOURCEID |
| RECORDID | Long | RECORDID |

**BIBREF DATA STRUCTURE**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name (Table 3.8.4)** |
| REFID | long | REFID |
| PUBTYPE | long | PUBLICATION\_TYPE |
| PUBDATE | long | PUBLICATION\_DATE |
| AUTHORS | Char[100] | AUTHOR\_NAME |
| EDITORS | Char[100] | EDITOR\_NAME |
| ANALYT | Char[255] | ANALYT |
| MONOG | Char[255] | MONOG |
| SERIES | Char[255] | SERIES |
| VOLUME | Char[10] | VOLUME |
| ISSUE | Char[10] | ISSUE |
| PAGECOL | Char[25] | PAGECOL |
| PUBLISH | Char[50] | PUBLISH |
| PUBCITY | Char[30] | PUBCITY |
| PUBCNTRY | Char[75] | PUBCNTRY |

**GEMS DATA STRUCTURE**

**GEMS\_INFO structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| GOBJID | LONG | GOBJID |
| GNVAL | CHAR[256] | GNVAL |
| GOBJTYPE | CHAR[256] | GOBJTYPE |
| MARKERID | LONG | MARKERID |

**GEMS\_MARKER\_DETECTOR structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| MDID | LONG | MDID |
| MATYPE | CHAR[256] | MATYPE |
| FPRIMER | CHAR[256] | FPRIMER |
| RPRIMER | CHAR[256] | RPRIMER |
| LMID | LONG | MARKERID |
| MAUID | LONG | MAUID |
| MAREF | LONG | MAREF |
| MINALLELE | LONG | MINALLELE |
| MAXALLELE | LONG | MAXALLELE |

**GEMS\_MV structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| MVID | LONG | MVID |
| MARKERID | LONG | MARKERID |
| MVTYPE | LONG | MVTYPE |
| MWT | DOUBLE | MWT |
| MDID | LONG | MDID |
| LMVID | LONG | LMVID |
| MVUID | LONG | MVUID |
| MVREF | LONG | MVREF |

**GEMS\_NAME structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| GNID | LONG | GNID |
| GOBJID | LONG | GOBJID |
| GOBJTYPE | CHAR[256] | GOBJTYPE |
| GNTYPE | LONG | GNTYPE |
| GNSTAT | LONG | GNSTAT |
| GNUID | LONG | GNUID |
| GNVAL | CHAR(256) | GNVAL |
| GNLOCN | LONG | GNLOCN |
| GNDATE | LONG | GNDATE |
| GNREF | LONG | GNREF |

**GEMS\_COMPONENT structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| CID | LONG | CID |
| CONDID | LONG | CONDID |
| COMID | LONG | COMID |
| COMVAL | CHAR(256]) | GNTYPE |
| PID | LONG | PID |
| COMTYPE | CHAR(256) | COMTYPE |
| COMGRP | CHAR(256) | COMGRP |
| COMUID | LONG | COMUID |
| COMREF | LONG | COMREF |

**GEMS\_PD\_COMP structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| PD\_COMP | LONG | PD\_COMP |
| PDID | LONG | PDID |
| CID | LONG | CID |

**GEMS\_PROP structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| PID | LONG | PID |
| PROPID | LONG | PROPID |
| PROPNAME | CHAR(256) | PROPNAME |
| SCALEID | LONG | SCALEID |
| METHID | LONG | METHID |
| PROPGRP | CHAR(256) | PROPGRP |

**GEMS\_PD structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| PDID | LONG | PDID |
| CONDID | LONG | CONDID |
| MDID | LONG | MDID |

**GEMS\_CONDITION structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| CONID | LONG | CONID |
| CONDNAME | CHAR(256) | CONDNAME |

**PD\_COMP\_INFO structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| CID | LONG | CID |
| CONDITION | LONG | CONDITION |
| PDID | LONG | PDID |

**GEMS\_SCALE structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| SCALEID | LONG | SCALEID |
| SCNAME | CHAR(256) | SCNAME |
| PROPID | LONG | PROPID |
| SCTYPE | CHAR(256) | SCTYPE |

**GEMS\_METHOD structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| METHID | LONG | METHID |
| MNAME | CHAR(256) | MNAME |
| MABBR | CHAR(256) | MABBR |
| MDESC | CHAR(256) | MDESC |

**PROP\_INFO structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| PID | LONG | PID |
| MNAME | CHAR(256) | MNAME |
| SNAME | CHAR(256) | SNAME |
| PNAME | CHAR(256) | PNAME |
| PGRPNAME | CHAR(256) | PGRPNAME |

**GEMS\_LOCUS structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| LOCUSID | LONG | LOCUSID |
| CHROMOSOME | CHAR(256) | CHROMOSOME |
| POSITION | CHAR(256) | POSITION |

**SUPPORTING TABLES**

**SUP\_PERSONS structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| PERSONID | LONG | PERSONID |
| FNAME | CHAR[24] | FNAME |
| LNAME | CHAR[52] | LNAME |
| IONAME | CHAR[16] | IONAME |
| INSTITID | long | INSTITID |
| PTITLE | CHAR[28] | PTITLE |
| PONAME | char[52] | PONAME |
| PLANGU | LONG | PLANGU |
| PPHONE | CHAR[24] | PPHONE |
| PEXTENT | CHAR[24] | PEXTENT |
| PFAX | CHAR[24] | PFAX |
| PEMAIL | CHAR[44] | PEMAIL |
| PROLE | LONG | PROLE |
| SPERSON | LONG | SPERSON |
| EPERSON | LONG | EPERSON |
| PSTATUS | LONG | PSTATUS |
| CONTACT | CHAR[256] | CONTACT |

**SUP\_INSTITUTE structure**

|  |  |  |
| --- | --- | --- |
| **Element name** | **Type** | **Database Field Name** |
| INSTITID | LONG | INSTITID |
| PINSID | LONG | PINSID |
| INSNAME | CHAR[152] | INSNAME |
| INSACR | CHAR[24] | INSACR |
| INSTYPE | LONG | INSTYPE |
| STREET | char[128] | STREET |
| POSTBOX | CHAR[28] | POSTBOX |
| CITY | CHAR[36] | CITY |
| STATEID | LONG | STATEID |
| CPOSTAL | CHAR[12] | CPOSTAL |
| CNTRYID | LONG | CNTRYID |
| APHONE | char[28] | APHONE |
| AFAX | CHAR[28] | AFAX |
| AEMAIL | CHAR[44] | AEMAIL |
| WEBURL | CHAR[64] | WEBURL |
| SINS | LONG | SINS |
| EINS | LONG | EINS |
| ICHANGE | LONG | ICHANGE |

**DLL Functions for Access Control**

Access to the different subsystems of ICIS is through a pair of open and close commands. Before any functions associated to the subsystem can be called, the open access command must executed first. However, access to the ICIS applications is based on the right granted to the current user of the installation as specified in the USERS and INSTLN tables. This right is checked and verified through the open access function of GMS subsystem.

**GENEALOGY MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS**

**GMS\_openDatabase**

GMS\_openDatabase reads the initialization file identified by the value of argument szIniFile. The data source names, DBMS user IDs and DBMS passwords are read from this file and the associated databases opened. GMS\_openDatabase does not check ICIS User IDs or passwords, but logs the user as GUEST with password GUEST with read-only access to the central and local database. To obtain full access to either database, use GMS\_openDatabase2. The format of the INI file is described in 5.2.3b, and the convention is to set szIniFile to ‘ICIS.INI’, which should exist in the default windows %TEMP% directory (6.1.1). It is not necessary to follow this convention for specific applications.

**Syntax**

long **GMS\_openDatabase**(*szIniFile*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| szIniFile | char \* | Input | Pointer to NULL terminated string containing the name of an existing initialization file. If an INI file other than %TEMP%\ICIS.INI is used, it should be passed as a parameter to the application including the full pathname. |

**Returns**

GMS\_SUCCESS, GMS\_ERRORor GMS\_NOACCESS

**Comments**

The function returns GMS\_ERROR if the one or both GMS databases are not opened successfully. No data sources are connected unless all are found and opened.

**GMS\_openDatabase2**

GMS\_openDatabase2 reads the initialization file identified by the value of argument szIniFile. There are two levels of access control to ICIS databases. First, the database management system may have a system of user IDs and passwords to control opening the database. Secondly, GMS has a table of user IDs, access privileges, and passwords controlling access to the ICIS fundamental functions. The data source names, database user IDs and database passwords are read from the initialization file szIniFile. The ICIS user name of the current user is passed in szUserName and the ICIS password in szPassword. These must be obtained by the application, and one short cut to this is to read them from the INI fil. The function returns the USER\_ID of the current user in argument userID.

**Syntax**

long **GMS\_openDatabase2**(*szIniFile, szUserName, szPassword, userID*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| szIniFile | char \* | input | Pointer to NULL terminated string containing the name of an existing initialization file. The file must exist and be present in the Windows subdirectory. |
| szUserName | char \* | input | Pointer to NULL terminated string containing the user name |
| szPassword | char \* | input | Pointer to NULL terminated string containing the password of the user |
| userID | long \* | output | Pointer to a long integer to contain the User ID corresponding to szUserName or –32678 for a guest user. |

**Returns**

GMS\_SUCCESS, GMS\_ERROR GMS\_NO\_ACCESS, GMS\_INVALID\_PASSWORD, GMS\_INVALID\_USER, or GMS\_INVALID\_INSTALLATION .

**Comments**

The function returns GMS\_INVALID\_USER if szUserName is not found. If the szPassword is incorrect it returns GMS\_INVALID\_PASSWORD. If the user’s installation (USERS.INSTALID) is zero (database administrators only) then the function will allow write access to any local database. Otherwise, the function checks that the local installation (INSTLN.INSTALID) is the same as the user’s installation (USERS.INSTALID). If the user’s installation is not zero and does not match the current local database, the function restricts the user to read only access and returns GMS\_READ\_ONLY.

If the local installation (INSTLN.INSTALID) is negative (new installation) then no further checks are made, but if it is positive (existing installation) then the function checks that the local installation fields are identical to the central fields for the matching installation. If not, the function returns GMS\_INVALID\_INSTALLATION.

The function also checks the security setting specified in the SECURITY key of the ICIS.INI file. If the security setting is 1, the user and password in the INI file is used to open the database and is checked against . If the security setting is 2, access is allowed only to users with USTATUS = 2 and the password is encrypted to have access to ICIS database. . If the security setting is 3, access is allowed only to users with USTAUS = 3. Password is encrypted to have access to ICIS database and user name is encrypted to have access to central GMS Database. If the security setting is 4, access is allowed only to users with USTATUS = 4; password is encrypted to have access to RDBMS database and to ICIS DLL.

The function returns GMS\_ERROR if the one or both GMS databases are not opened successfully, or cause by any internal error.

**GMS\_closeDatabase**

GMS\_closeDatabase closes the connection to both Central and Local databases. Frees up all allocated memory used by the databases and DLL.

**Syntax** (Procedure)

GMS\_closeDatabase

**Arguments**

No arguments

**Returns**

No return value

**GMS\_commitData**

GMS\_commitData will permanently write all changes to the database made by calling add or change functions.

**Syntax**

**GMS\_commitData** No arguments, No returns

**Comment**

Caution should be observed in using this function because it clears the cache used by the SQL driver and clears all SQL transactions. Hence if it is called between a FIND\_FIRST and FIND\_NEXT sequence in a search, a ‘function sequence error’ will occur on the next search call with fOpt set to FIND\_NEXT.

**DATA MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS**

**DMS\_openDatabase**

DMS\_openDatabase is the primary function that establishes connection to the DMS databases through the initialization file identified by the argument szInifile. Data source names, DBMS user IDs and pass words needed to connect are read from this file.

**Syntax**

long DMS\_openDatabase(szInifile)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| szInifile | Char \* | Input | Pointer to a NULL terminated string containing the name of an existing initialization file. This file must reside in the default WINDOWS directory. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_closeDatabase**

DMS\_closeDatabase closes connection to both Central and Local DMS databases. The function also frees up allocated memory of handlers associated with it.

**Syntax** (Procedure)

void **DMS\_closeDatabase**()

**Arguments**

No arguments

**Returns**

No return value

**DMS\_commitData**

DMS\_commitData will permanently write all changes to the database made by calling add or change functions.

**Syntax**

**DMS\_commitData** No arguments, No returns

**Comment**

Caution should be observed in using this function because it clears the cache used by the SQL driver and clears all SQL transactions. Hence if it is called between a FIND\_FIRST and FIND\_NEXT sequence in a search, a ‘function sequence error’ will occur on the next search call with fOpt set to FIND\_NEXT.

**INVENTORY MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS**

**IMS\_OpenDatabase**

IMS\_OpenDatabase is the primary function that establishes a connection to the inventory database through the initialisation file identified by the argument szInifile. Data source names, DBMS user IDs and passwords are read from this file.

**Syntax**

long IMS\_OpenDatabase(szInifile)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| szInifile | Char | Input | Pointer to a NULL terminated string containing the name of an existing initialisation file. This file must reside in the default WINDOWS directory. |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1)

**IMS\_CloseDatabase**

IMS\_closeDatabase closes connection to the inventory database. The function also frees up allocated memory used by the DLL.

**Syntax**

void IMS\_closeDatabase()

**Arguments**

No arguments

**Returns**

No return value

**IMS\_commitData**

IMS\_commitData will permanently write all changes to the IMS database made by calling add or change functions.

**Syntax**

**IMS\_commitData** No arguments, No returns

**Comment**

Caution should be observed in using this function because it clears the cache used by the SQL driver and clears all SQL transactions. Hence if it is called between a FIND\_FIRST and FIND\_NEXT sequence in a search, a ‘function sequence error’ will occur on the next search call with fOpt set to FIND\_NEXT.

**GENE MANAGEMENT SYSTEM DATABASE ACCESS FUNCTIONS**

**GEMS\_openDatabase**

GEMS\_openDatabase establishes connection to the GEMS database using the information specified in the initialization file identified by the argument szInifile. The information read from this file are Data source names, DBMS user IDs and pass words needed to connect.

**Syntax**

long GEMS\_openDatabase(szInifile)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| szInifile | Char \* | Input | Pointer to a NULL terminated string containing the name of an existing initialization file. This file must reside in the default WINDOWS directory. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**GEMS\_closeDatabase**

GEMS\_closeDatabase closes connection to GEMS database. The function also frees up allocated memory of handlers associated with it.

**Syntax** (Procedure)

void **GEMS\_closeDatabase**()

**Arguments**

No arguments

**Returns**

No return value

**GEMS\_commitData**

GEMS\_commitData will permanently write all changes to the GEMS database made by calling add or change functions.

**Syntax**

**GEMS\_commitData** No arguments, No returns

**Comment**

Caution should be observed in using this function because it clears the cache used by the SQL driver and clears all SQL transactions. Hence if it is called between a FIND\_FIRST and FIND\_NEXT sequence in a search, a ‘function sequence error’ will occur on the next search call with fOpt set to FIND\_NEXT.

**USER AND INSTALLATION**

**GMS\_getUser**

GMS\_getUser will retrieve user information. The first argument is a pointer to a GMS\_User structure (6.1.2g). If the USER\_ID element is not zero when the function is called then it returns the details for the corresponding user. If it is zero then a user name must be supplied in the NAME element and the function returns details for the first user with the specified name if fOpt = FIND\_FIRST or for the next available user with that name if fOpt = FIND\_NEXT. Any call which fails to get user information returns GMS\_NO\_DATA.

**Syntax**

long GMS\_getUser(user, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| User | GMS\_User\* | input/output | Address of a GMS\_User structure |
| fOpt | Long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The principle of using GMS\_getUser is almost the same as GMS\_getAttribute. If there is a problem accessing the database, the function will return GMS\_ERROR.

**GMS\_getInstallation**

GMS\_getInstallation will retrieve installation information.

**Syntax**

long GMS\_getInstallation(instln,szDesc,nszDesc)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| instln | GMS\_Installation\* | input/ output | Address of a GMS\_Installation structure to contain the installation parameters. |
| szDesc | char\* | output | Address of a character variable to contain the installation description |
| nszDesc | Long | input | Maximum data size szDesc can hold (including the null terminator). |

**Returns**

GMS\_ERROR, GMS\_NO\_DATA , GMS\_SUCCESS

**Comments**

If the instno field of instln structure is zero then the function will return the current installation record from the local database. Otherwise, it will return the installation record corresponding to the value of the instno field. The function returns GMS\_ERROR if there is no local INSTALLATION table or if the installation record corresponding to INSTNO field is not found in the central database. Otherwise, it returns GMS\_SUCCESS .

**GMS\_addUser**

GMS\_addUser adds a new user to the local USERS table. Details of the reference must be stored in appropriate elements of the user parameter. The next available user number will be returned in the USERID element. New data is not committed to the local database until the function GMS\_commitData (6.8.1) has been called, and uncommitted data can be deleted from the buffer with the GMS\_rollbackData function (6.8.2)

**Syntax**

long GMS\_addUser(user,szEname,security)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| user | GMS\_USERS \* | input/output | Address of a GMS\_BIBREFS structure |
| szEname | Char \* | output | Address of NULL terminated string to place the encrypted user name |
| security | long | input | The level of security |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**GMS DLL FUNCTIONS**

**SEARCH FUNCTIONS**

**GMS\_findName**

GMS\_findName searches all names in the NAMES tables for the search string szNameSrch. If fOpt is FIND\_FIRST the function stores the name details of the first match of the germplasm record that has not been replaced in structure nameData and the NAME value in szName. The germplasm details of the matched record are retrieved to the structure germData.. If fOpt is FIND\_NEXT it stores the succeeding non-replaced match.

**Syntax**

long **GMS\_findName**(*szNameSrch, nameData, germData, szName, nszName, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| SzNameSrch | char \* | input | Address of NULL terminated string containing the name search |
| NameData | GMS\_NameData\* | input/ output | Address of a GMS\_NameData structure to contain the name data |
| GermData | GMS\_Germplasm\* | output | Address of a GMS\_Germplasm Structure to contain Germplasm data for the retrieved germplasm |
| SzName | char \* | output | Address of NULL terminated string to place the NAME |
| NszName | long | input | Maximum length of szName |
| Fopt | long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The target name can contain wildcards within the 128 characters of the string. ‘\_’ match any character and a terminal ‘%’ matches any trailing characters. On the first call to this function, fOpt should have a value of FIND\_FIRST then call the function with fOpt set to FIND\_NEXT repeatedly until it returns GMS\_NO\_DATA to find subsequent matching names.

The function updates the name record based on the modification stored in the CHANGES table.

**Returns**

GMS\_ERROR if any internal ODBC function call fails.

**GMS\_findName2**

GMS\_findName2 searches for string matches to any of three forms of a specified name. The forms of the name are the specified string, the string with all spaces removed and with name standardization applied (see also the [GMS\_standardName](file:///E:\ICIS5\iciswiki\articles\g\m\s\GMS_standardName_1cd0.html) function ). The function returns name and germplasm data for search matches. If the STATUS field of the NameData argument is non-zero then only names of the specified status are returned.

**Note**

* This routine calls [GMS\_applyChanges](file:///E:\ICIS5\iciswiki\articles\g\m\s\GMS_applyChanges_8f03.html) internally to update the germplasm record.
* The function updates the name record based on the modification stored in the CHANGES table.

**Syntax**

long **GMS\_findName2** (*szNameSrch, nameData, germData, szName, nszName, fOpt*)

**Arguments**

As for GMS\_findName except that a new argument Germ is included:

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| szNameSrch | char \* | input | Address of NULL terminated string containing the name search |
| NameData | GMS\_NameData\* | input/ output | Address of a GMS\_NameData structure to contain the name data |
| GermData | GMS\_Germplasm\* | output | Address of a GMS\_Germplasm Structure to contain Germplasm data for the retrieved germplasm |
| SzName | char \* | output | Address of NULL terminated string to place the NAME |
| NszName | long | input | Maximum length of szName |
| Fopt | long | input | Either FIND\_FIRST or FIND\_NEXT |

**Comments**

This function works like GMS\_findName but searches for matches to the supplied name szNameSrch with any internal spaces removed and after applying the name standardization algorithm (3.1.3 and 6.9.8). Wildcards characters should appear within the first 128 characters of the strings with spaces removed and after name standardization, otherwise they will not be recognized as wildcards.

The function updates the name record based on the modification stored in the CHANGES table.

**GMS\_findName3**

GMS\_findName3 searches for a string in the NAMES table, both as specified and with any spaces removed, and returns name and germplasm data for search matches. A value in TYPE or STATUS or LOCATION is specified, only those records matching the combination of values are returned. Otherwise, it functions like GMS\_findName.

**Note**

This routine calls GMS\_applyChanges (6.4.10) internally to update the germplasm record before returning.

**Syntax**

long **GMS\_findName3** (*szNameSrch, nameData, germData, szName, nszName, fOpt*)

**Arguments**

As for GMS\_findName except that a new argument GermData is included:

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| GermData | GMS\_Germplasm\* | output | Address of a GMS\_Germplasm Structure to contain Germplasm data for the retrieved germplasm |

**Comments**

This function works like GMS\_findName but also returns the germplasm record and also searches for matches to the supplied name szNameSrch with any internal spaces removed but does not apply the name standardization algorithm.

The function updates the name record based on the modification stored in the CHANGES table.

Also see GMS\_findName comments.

GMS\_ERROR if any internal ODBC function call fails.

**GMS\_findDescendant**

GMS\_findDescendant searches the GPID1 and GPID2 fields of the GERMPLASM Table and the PID field of the OTHER PROGENITORS Table for the given *gid*. After a successful call to the function, *descendant* will contain the germplasm record of a derivative or offspring of the target germplasm and *progNo* will contain the number of the progenitor that matches *GID*.

**Note**

This routine calls GMS\_applyChanges (6.4.10) internally to update the germplasm record before returning.

**Syntax**

long **GMS\_findDescendant**(gid, descendant, progNo, fOpt);

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Gid | long | input | GID for which descendants are sought |
| descendant | GMS-germplasm\* | output | Pointer to a Germplasm record which will contain the germplasm data for a descendant of GID |
| progNo | long\* | output | Number of the progenitor which matches GID |
| fOpt | long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Note**

GMS\_findDescendant checks whether progenitor progNo has been changed through a local attribute before returning a value and it checks for progenitors which have been changed to gid by local change attributes.

**GMS\_findGermplasm**

GMS\_findGermplasm will retrieve germplasm records for all germplasm matching specified criteria. The first argument of GMS\_findGermplasm is a pointer to a GMS\_Germplasm structure. All elements of the structure must be set to valid values or zeros before calling the function with parameter fOpt=FIND\_FIRST. This sets a search template restricting the search to records with fields matching the valid values. The first match is returned if *fOpt* = FIND\_FIRST and the next matching location is returned if *fOpt* = FIND\_NEXT

**Syntax**

long **GMS\_findGermplasm**(*germData, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| germData | GMS\_Germplasm \* | Input/ output | Enters with the name-string to search, returns with the location details of the next match |
| fOpt | Long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GMS\_findCIDSID**

GMS\_findCIDSID searches for ICIS GID, CIMMYT CID and SID depending on the given parameters. If CID and/or SID codes are given, it retrieves the corresponding GID. If GID is given, CID and SID are returned. FIND\_FIRST and FIND\_NEXT are valid find options if CID is the input parameter. If CID and SID are provided, the find option should be FIND\_SPECIFIC=2. If GID is the input parameter, FIND\_FIRST option is used.

**Syntax**

long **GMS\_findCIDSID***(CID,SID,GID, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| CID | long | Input/ output | The CID to search if input parameter or the CID returned by the function if GID is given. |
| SID | long | Input/ output | The SID to search if input parameter together with CID or the SID returned by the function if GID or CID is given. |
| GID | long | Input/ output | The GID to search if input parameter or the GID returned by the function if CID and/or SID are given. |
| fOpt | long | input | Either FIND\_FIRST(0) or FIND\_NEXT(1) if CID is given; FIND\_FIRST(0) if GID is given; FIND\_SPECIFIC(2) if CID and SID are given. |

Note:

If the GERMPLSM table being accessed does not have CID and SID, the function returns GMS\_INVALID\_DBSTRUCTURE;

**GMS DATA RETRIEVAL functions**

**GMS\_getGermplasmRecord**

GMS\_getGermplasmRecord retrieves data from the GERMPLASM table with GERMPLASM\_ID equal to *gid* in structure *germData.* The user must check the return status and the REPLACE field of *germData* for replaced or deleted germplasm and take appropriate action. No check is made in the local database for change or replace attributes. These can be applied by calling **GMS\_applyChanges** (6.4.10).

**Syntax**

long GMS\_getGermplasmRecord (*gid*, *germData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| gid | long | input | The GERMPLASM\_ID to search |
| germData | GMS\_Germplasm \* | output | Address of a GMS\_Germplasm record |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The GERMPLASM tables will be searched for the record with GERMPLASM\_ID equal to *gid*. It is not possible for the search to have multiple results because the GERMPLASM\_ID is unique across the two data sources (Central and Local GMS). It returns GMS\_NO\_DATA if there is no GERMPLASM record with GERMPLASM\_ID (*gid*). If the record has been deleted or replaced (GRPLCE=*gid* for deleted, GRPLCE= <*replacement gid*> for replace) the function returns the deleted or replaced germplasm record and GMS\_NO\_DATA. It returns GMS\_ERROR if any internal ODBC function call fails.

**GMS\_getGermplasm**

GMS\_getGermplasm retrieves data from the GERMPLASM table with GERMPLASM\_ID equal to *gid* in structure germData.

**Syntax**

long **GMS\_getGermplasm** (*gid*, *germData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| gid | long | input | The GERMPLASM\_ID to search |
| germData | GMS\_Germplasm \* | output | Address of a GMS\_Germplasm record |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Note**

This routine calls GMS\_applyChanges (6.4.10) internally to update the germplasm record before returning.

**Comments**

See GMS\_getGermplasmRecord comments.

**GMS\_getGermplasm2**

GMS\_getGermplasm2 retrieves all data from the GERMPLASM table and data for the preferred name from the NAMES table for germplasm *gid* or for its replacement if it is replaced. The function does not return data if germplasm *gid* is deleted. The germplasm data is returned in argument *germData,* the name data in argument *nameData* and the preferred name in *szName*. With GMS\_getGermplasm2 the user only needs to check the return status for deleted germplasm.

**Note**

GMS\_getGermplasm2 checks the local CHANGES table for any replace or change attributes by calling GMS\_applyChanges internally. If it finds them it applies the specified deletion, replacement or change before returning the data. This means that a local user can specify changes to central records and all applications that use this function will show the effect of those changes.

The function updates the name record based on the modification stored in the CHANGES table.

**Syntax**

long **GMS\_getGermplasm2** (gid, germData, nameData, szName, nszName)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| gid | long | input | The GERMPLASM\_ID to search |
| germData | GMS\_Germplasm \* | output | Address of a GMS\_Germplasm record |
| nameData | GMS\_NameData\* | output | Address of a GMS\_NameData structure to contain details of the preferred name. |
| szName | char\* | output | Address of a string variable to contain the preferred name as a null terminated string |
| nszName | long | input | Maximum length of the *szName* |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

If germplasm *gid* has been deleted (GRPLCE=GID), GMS\_getGermplasm2sets the GERMPLASM\_ID element of argument *germData* to zero and returns GMS\_NO\_DATA. If the germplasm *gid* has been replaced (GRPLCE= *<replacement gid>*) the function returns data for the replaced germplasm and not germplasm GID. In this case the GERMPLASM\_ID element of *germData* will have a different value to *gid*. The routine also checks the local ATTRIBUTES table for any change or replacement attributes for the germplasm record and if it finds then the function applies the changes. If there is an ODBC error the function returns GMS\_ERROR.

**GMS\_getGermplasm3**

GMS\_getGermplasm3 works exactly like GMS\_getGermplasm2 but with additional return argument for the abbreviation.

**Note**

See GMS\_getGermplasm2 note.

**Syntax**

long **GMS\_getGermplasm3** (gid, germData, nameData, szName, nszName, szAbbreviation)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| gid | long | input | The GERMPLASM\_ID to search |
| germData | GMS\_Germplasm \* | output | Address of a GMS\_Germplasm record |
| nameData | GMS\_NameData\* | output | Address of a GMS\_NameData structure to contain details of the preferred name. |
| szName | char\* | output | Address of a string variable to contain the preferred name as a null terminated string |
| nszName | long | input | Maximum length of the *szName* |
| szAbbreviation | char\* | output | Address of a string variable to contain the preferred abbreviation name as a null terminated string |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

GMS\_getGermplasm3 does not specify the length of the *szAbbreviation* for the routine expects it to be at least 9 bytes in length (>=9).

See GMS\_getGermplasm2 comments.

**GMS\_getProgenitorID**

GMS\_getProgenitorID retrieves theGERMPLASM\_ID of *iProgen*th progenitor of germplasm with GERMPLASM\_ID *gid* in variable *pGid*.

**Syntax**

long **GMS\_getProgenitorID**( gid, iProgent, pGid)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| gid | long | input | The GERMPLASM\_ID to search |
| iProgen | long | input | No of the parent whose ID is required |
| pGid | long \* | output | Address of a long integer variable to contain the GERMPLASM\_ID of the *iProgen*th progenitor |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

Returns GMS\_NO\_DATA when there is no *iProgen*th progenitor. Returns GMS\_ERROR if any internal ODBC function call fails.

Although GMS\_getProgenitorID can be called with *iProgen* equal to 1 or 2 it is usually not necessary as the germplasm data containing these parent IDs is often already available, and if not it is still more efficient to call it from the GMS\_getGermplasm function. The routine checks the local ATTRIBUTES table for any changes in *iProgen*th progenitor.

See GMS\_getGermplasm2 comments.

**GMS\_getNameRecord**

GMS\_getNameRecord retrieves data from the NAMES table for the given *nid* in structure *nameData.* The user must check the return NSTAT field of *nameData* for deleted name and take appropriate action.

**Syntax**

long GMS\_getNameRecord (*nameData,szName,nszName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| nameData | GMS\_Name \* | Input/output | Address of a GMS\_Name record |
| szName | Char \* | output | Address of a string variable to contain the name as a null terminated string |
| nszNamr | long | input | The length of the szName string. |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GMS\_getName**

GMS\_getName will retrieve all names associated with germplasm having specified germplasm ID and name STATUS or name TYPE. GMS\_getName requires that the target germplasm ID is stored in the GERMPLASM element of the *nameData* argument (6.1.2b) and the STATUS and TYPE elements should be set to zero or a desired value before calling the function. If TYPE and STATUS are zero, the function will retrieve all names of the target germplasm. If STATUS is non zero only names with matching status values are returned. If STATUS is zero and TYPE is non zero then only names with matching TYPE are returned. If *fOpt* equals FIND\_FIRST the function returns the Name with the lowest value of TYPE. Subsequent calls with *fOpt* equal to FIND\_NEXT return names in order of value of TYPE. Any call that fails to find a name returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getName**(nameData, szName, nszName, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| nameData | GMS\_NameData\* | input/output | Address of a GMS\_NameData record |
| szName | char \* | output | Address of NULL terminated string containing the NAME |
| nszName | long | input | Maximum size of *szName* in bytes |
| fOpt | long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The germplasm ID and, if required, the name type of the germplasm for which names are required must be loaded into the GERMPLASM and TYPE elements of the GMS\_NameData argument if fOpt=FIND\_FIRST. On return this argument contains all the other relevant fields for the returned name. If no name is found the argument *szName* will contain a NULL string. If the function is called with bad parameters or there was a problem accessing the database, the function will return GMS\_ERROR. GMS\_getName returns name data for deleted or replaced germplasm without warning.

**GMS\_getAttribute**

GMS\_getAttribute will retrieve all attributes associated with germplasm having specified germplasm ID and attribute TYPE. GMS\_getAttribute requires the germplasm ID and attribute type are stored in the GERMPLASM and TYPE fields of the *attribute* argument before calling the function. If TYPE is zero the function will retrieve all attributes of a germplasm regardless of attribute type. If *fOpt* equals FIND\_FIRST the function returns the attribute with the lowest value of attribute TYPE. Subsequent calls with *fOpt* equal to FIND\_NEXT return attributes in order of TYPE value. Any call that fails to find an attribute returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getAttribute**(attribute, szAVal, nszAVal, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| attribute | GMS\_Attribute\* | input/output | Address of a GMS\_Attribute record |
| szAVal | char \* | output | Address of NULL terminated string to place the attribute value |
| nszAVal | Long | input | Length/Size of *szAVal* |
| fOpt | Long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The principle of using GMS\_getAttribute is almost the same as GMS\_getName. The only difference is that the first argument is GMS\_Attribute structure and the ATTRIBUTE table is searched. If the function is called with bad parameters or there was a problem accessing the database, the function will return GMS\_ERROR. GMS\_getAttribute returns attribute data for deleted or replaced germplasm without warning.

**GMS\_getMethod**

GMS\_getMethod retrieves details of a method from the METHODS table. The first argument of GMS\_getMethod is a pointer to a GMS\_Method structure (6.1.2d). If the METHOD\_ID element is not zero when the function is called then the details for the corresponding method are retrieved. If it is zero then the function returns details for the first method if *fOpt* = FIND\_FIRST or for the next available method if *fOpt* = FIND\_NEXT. Any call which fails to find a method information returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getMethod**(method, szMDesc,nszMDesc, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| method | GMS\_Method\* | input/ output | Address of a GMS\_Method structure to contain method details |
| szMDesc | char \* | output | Address of a string variable to contain the NULL terminated method description |
| nszMDesc | Long | input | Maximum size of *szMDesc* in bytes |
| fOpt | Long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The principle of using **GMS\_getMethod** is almost the same as **GMS\_getAttribute**. If the function is called and there was a problem accessing the database, the function will return GMS\_ERROR. If *szMDesc* is NULL and *cMDesc* is zero then GMS\_getMethod will not retrieve the method description.

**GMS\_applyChanges**

GMS\_applyChanges returns the germplasm record after applying the changes as recorded in the local attributes table. This allows a local user to register changes to central germplasm and have those changes reflected immediately in all analysis applications that use the DLL. The changes are checked and physically made in the central database at update.

**Syntax**

long **GMS\_applyChanges**(*germData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| GermData | GMS\_Germplasm \* | input/ output | Address of a GMS\_Germplasm record |

**Returns**

GMS\_ERROR,GMS\_SUCCESS

**Comments**

GMS\_applyChanges searches the local ATTRIBUTS table where GID matches *germData.gid* and ATYPE value is either REPLACE (101) or CHANGE(102). GMS\_applyChanges first looks for REPLACE attributes and gets the replacement germplasm. Next it looks for CHANGE attributes and applies the changes in chronological order with the most recent change last. GMS\_applyChanges is called internally for central germplasm (positive gids) by the following routines before returning the germplasm data: GMS\_getGermplasm2, GMS\_getGermplasm3, GMS\_getDescendant, GMS\_findName, GMS\_findName2, and GMS\_findName3 GMS\_findPreferredName**.**

**Functions to add data to the local database**

**GMS\_addGermplasm**

GMS\_addGermplasm will add new germplasm and name information to the GMS database according to the information in the arguments. If the germplasm is produced by a generative method with more than 2 progenitors then germplasm IDs for progenitors after the second are contained in array argument *aPID*.

The GERMPLASM\_ID field of the *germData* argument and the GERMPLASM field of the *nameData* argument must be zero on input, and will be set to the new germplasm ID value on exit.

New data is not committed to the local database until the function GMS\_commitData (6.8.1) has been called, and uncommitted data can be deleted from the buffer with the GMS\_rollbackData function (6.8.2)

**Syntax**

long **GMS\_addGermplasm**( germData, nameData, szPName, aPID)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| germData | GMS\_Germplasm \* | input/ output | Address of a GMS\_Germplasm record |
| nameData | GMS\_NameData \* | input/ output | Address of a GMS\_NameData record |
| szPName | char \* | input | Address of NULL terminated string which contains the PREFERRED\_NAME |
| *aPID* | long \* | input | Address of an array of Parent GIDs. NULL if the number of progenitors is less than or equal to two. |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_ACCESS

**Comments**

The following fields of the *germData* structure should be properly populated with valid values before calling the function:

• METHOD – number of method used corresponding to METHOD\_ID on the METHODS table;

• NO\_PROGENITORS - for a derivative process or the number of parents for a generative process;

• PROGENITOR\_ID1 - Group ID for a derivative process or the ID of the first parent (usually the female) for a generative process. (Zero if unknown);

• PROGENITOR\_ID2 - Source ID for a derivative process or the ID of the second parent for a generative process. (Zero if unknown);

• LOCATION - Location ID for the breeding or collection location (Zero if unknown);

• DATE - Date as YYYMMDD ( 0 if unknown);

• REFERENCE – reference number (missing id unknown).

The elements GERMPLASM\_ID, USER and REPLACE will be populated by the function.

The string *szPname* may not be NULL and the following fields of the *Name* structure should be populated with valid values before calling the function:

• TYPE – name type number from the UDFIELDS table;

• STATUS - name status (preferred or not) and data storage mode (Table 3.8.3).

• LOCATION – name location number, often the same as LOCATION in *Germ*;

• DATE - date the germplasm was named, same form as DATE in *Germ*;

• REFERENCE - if there is one available.

The GERMPLASM and USER elements of *Name* will be populated by the function.

**GMS\_addName**

GMS\_addName adds new name *szPName* for germplasm specified by the GERMPLASM field in the *nameData* structure.

New data is not committed to the local database until the function GMS\_commitData (6.8.1) has been called, and uncommitted data can be deleted from the buffer with the GMS\_rollbackData function (6.8.2)

**Syntax**

long **GMS\_addName**(*nameData*, *szPName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| nameData | GMS\_NameData \* | input | Address of a GMS\_NameData record |
| szPName | Char \* | input | Address of NULL terminated string to place the name |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The string *szPname* may not be NULL and the following fields of the *Name* structure should be properly populated with valid values before calling the function:

• GERMPLASM – valid germplasm ID

• TYPE - the type of name;

• STATUS - name status (preferred or not) and data storage type;

• LOCATION

• DATE

• REFERENCE - if there is one available.

**GMS\_addAttribute**

GMS\_addAttribute adds new attribute with value *szAVal* for germplasm defined by the GERMPLASM\_ID field in the *attribute* structure.

New data is not committed to the local database until the function GMS\_commitData (6.8.1) has been called, and uncommitted data can be deleted from the buffer with the GMS\_rollbackData function (6.8.2)

**Syntax**

long **GMS\_addAttribute**(*attribute*, *szAVal*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| attribute | GMS\_Attribute \* | input/ output | Address of a GMS\_Attribute structure |
| szAVal | char \* | input | Address of NULL terminated string to place the value of the attribute |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The string *szAVal* should not be a NULL string and the following fields of the *attribute* structure should be properly populated with values before calling the function:

• GERMPLASM – valid germplasm ID

• TYPE - the type number of the attribute (from the UDFLDS Table);

• LOCATION

• DATE

• REFERENCE - if there is one available.

**GMS\_addChanges**

GMS\_addChanges adds one record to the local CHANGES table specifying a change made to germplasm data.

**Syntax**

long **GMS\_addChanges** (*changeData,szDesc*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *changeData* | GMS\_Changes \* | Input/output | Address of a GMS\_Changes structure |
| *szDesc* | char \* | Input | Description of the change |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**Comment**

The fields CTABLE, CFIELD, CRECORD, CFROM, CTO and CGROUP should be properly initialized. The routine will automatically assign values to CID, CUID, CDATE, and CTIME fields.

**Functions to correct LOCAL GMS records**

**GMS\_setGermplasm**

GMS\_setGermplasm changes the value of any field in the local GERMPLASM table except the PREFERRED\_NAME, NAME\_EXTENSION and ABBREVIATION fields. Only the central database administrator can change field values in the Central database. However, a local user can request such changes by adding SETGERM attributes for central germplasm records to the local attributes table.

**Syntax**

long **GMS\_setGermplasm**(*germData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *germData* | GMS\_Germplasm\* | Input | Pointer to a GMS-Germplasm structure containing correct values for all fields whether they are to be changed or not. |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**Comments**

This function can be used to correct certain fields in the local GERMPLASM table. A call with positive GID will result in a GMS\_ERROR return.

**GMS\_setProgenitorID**

GMS\_setProgenitorID changes the value of a progenitor ID for germplasm GID in the local GMS database. If argument *iProgen* has value 1 or 2, the value is changed on the germplasm record, otherwise it is changed in the PROGENITORS table.

**Syntax**

long **GMS\_setProgenitorID**(gid, iProgen,pGid)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *gid* | long | input | GID of germplasm for which progenitor ID is to be set |
| *iProgen* | long | input | Progenitor number to be set |
| *pGid* | long | input | GID to be stored as *iProgen*th progenitor of germplasm GID |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**Comments**

This function can be used to change the value of a progenitor ID either to another valid GID or to zero that indicates missing information on the progenitor. If *iProgen* is 1 or 2 the value could be changed with the GMS\_setGermplasm function, but GMS\_setProgenitorID also works, and is the only way to change the value if *iProgen* >2. If *gid* is positive the function returns GMS\_ERROR*.*

**GMS\_setPreferredName**

GMS\_setPreferredName sets an existing, ASCII name as the preferred name of a germplasm and resets the name\_status of the current preferred name to zero.

**Syntax**

long **GMS\_setPreferredName** (*nameData,szPName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *nameData* | GMS\_NameData\* | input | Pointer to a GMS\_NameData structure containing the name data of the existing name which is to be set as the preferred name |
| *szPName* | char\* | input | Pointer to a character variable containing the null terminated existing name to be set as the preferred name |

**Returns**

GMS\_SUCCESS , GMS\_ERROR

**Comments**

The new preferred name must exist on the NAMES table before the call to GMS-setPreferredName. The GERMPLASM\_ID, NAME\_USER, NVAL and NEXT fields should uniquely index the proposed preferred name. It must be an ASCII name, and not the current preferred name i.e. it must have NAME\_STATUS 0 or 2. Any other value results in an error return, GMS\_ERROR. If the proposed preferred name is already the preferred abbreviation then it is re-set as the preferred name. The NAME\_STATUS of the existing preferred name is changed to 0 on the NAMES table. If the GERMPLASM\_ID field of the *nameData* argument is positive the function returns GMS\_ERROR.

**GMS\_setPreferredAbbr**

GMS\_setPreferredAbbr sets an existing, ASCII name as the preferred abbreviation of a germplasm.

**Syntax**

long **GMS\_setPreferredAbbr**(*namedata,szName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *namedata* | GMS\_NameData\* | input | Pointer to a GMS\_NameData structure containing the name data of the existing name which is to be set as the preferred abbreviation |
| *szName* | char\* | input | Pointer to a character variable containing the null terminated existing name to be set as the preferred abbreviation |

**Returns**

GMS\_SUCCESS , GMS\_ERROR

**Comments**

The proposed preferred abbreviation must exist on the NAMES table before the call to GMS\_setPreferredAbbr. The GERMPLASM\_ID, NAME\_USER and NVAL fields should uniquely index the proposed preferred abbreviation. It must be an ASCII name of less than eight characters and it may not be the current preferred name or preferred abbreviation i.e. it must have NAME\_STATUS 0. Failure of these conditions results in an error return, GMS\_ERROR . The NAME\_STATUS of any existing preferred abbreviation is changed to 0 on the NAMES table. If the GERMPLASM\_ID field of the *namedata* argument is positive the function returns GMS\_ERROR.

**GMS\_setName**

GMS\_setName changes the value of any field in the local NAMES given the NID.

**Syntax**

long **GMS\_setName**(*nameData,szName,nszName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| nameData | GMS\_NameData \* | input | Pointer to a GMS-NameData structure containing correct values for all fields whether they are to be changed or not. |
| szName | Char \* | input | Address of NULL terminated string to place the name |
| nszName | Long | input | Maximum size of the szName string |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**Comments**

This function can be used to correct certain fields in the local NAMES table. A call with positive NID will result in a GMS\_ERROR return.

4.4.4 **GMS\_setAttribute**

GMS\_setAttribute changes the value of any field in the local ATTRIBUT table given the AID.

**Syntax**

long **GMS\_setAttribute**(*attrData,szAval*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| attrData | GMS\_Attribute\* | input | Pointer to a GMS\_Attribute structure containing correct values for all fields whether they are to be changed or not. |
| szAval | Char \* | input | Address of NULL terminated string to contain the attribute value |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**Comments**

This function can be used to correct certain fields in the local ATTRIBUT table. If an attribute is to be deleted, change the ATYPE = 999.

**BASIC GMS FUNCTIONS**

**GMS\_generateTree**

GMS\_generateTree retrieves preferred names and GIDs from the GERMPLASM table for all generative progenitors in the pedigree of germplasm *gid* down to a specified number of levels or until no further progenitors are known.

**Syntax:**

long **GMS\_generateTree**(gid,level,szBuffer,bufsz,aID,naID)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *gid* | long | input | Progenitor ID from which the tree will generate (as the root) |
| *level* | long | input | Desired maximum level of the tree |
| *szBuffer* | char \* | output | Points to a buffer that receives a string representing the genealogical tree. The buffer is filled with one or more null-terminated strings (representing one line on the screen); the last string is followed by a second null character. |
| *nszBuffer* | long | input | Specifies the size, in characters, of the buffer pointed to by the *szBuffer* parameter. |
| *aID* | long \* | output | Contains array of GIDs corresponding to the germplasm on each line of *szBuffer* |
| *naID* | long | input | Maximum number of entries in aID |

**Returns**

GMS\_SUCCESS ,GMS\_ERROR or GMS\_MORE\_DATA

**Comments**

*szBuffer* of size *nszBuffer* should be large enough to hold the whole generated tree. The routine will return GMS\_MORE\_DATA if the *szBuffer* does not contain the entire generated tree. Test the result for either GMS\_SUCCESS or GMS\_MORE\_DATA. Argument *aID* returns with the list of GIDs corresponding to each line of the tree so that extra information can be extracted from the database and displayed on the tree as required. If there are more lines (entries) in the tree than spaces in *aID* (*naID*) then **GMS\_generateTree** returns GMS\_MORE\_DATA.

The tree is displayed horizontally. A line starting with > indicates that the entry on the line above is a derivative of the entry on the current line. A line starting with + indicated that the entry on the current line is a generative progenitor of the entry above the +. Female progenitors are listed above male progenitors. \*\* following an entry name indicates that that entry has already been expanded in the current tree. < following an entry indicates that there is further pedigree information available on that entry.

**Code Example:**

The following code will display the generated tree:

**CODEEXAMPLE OUTPUT**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | char \*p; |  |  |  | IR8 | |
|  | p = szBuffer; |  |  |  | > IR8 | |
|  | | | while (\*p!=’\0’) |  |  | +--- PETA |
|  | | | | { // dump to the console |  | | > PETA |
|  | | | printf("s \n",p); |  |  | | +---CINA |
|  | | | | p = p + strlen(p)+1; |  | | +---LATISAIL |
|  | } |  |  |  |  | +--- DEE-GEO-WOO-GEN |

**GMS\_crossExpansion:**

GMS\_crossExpansion interrogates the GMS recursively to generate a Purdey cross expansion for germplasm *gid.* The expansion terminates after *level* generations or whenever a progenitor has a name of type *ntype*. The cross expansion is returned in argument *szBuffer*, and the list of GIDs for elements in the cross expansion (from left to right) is returned in array *aID*.

**Syntax**

long **GMS\_crossExpansion**(gid,level,ntype,szBuffer,nszBuffer,aID,naID)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *gid* | long | input | germplasm id |
| *level* | long | input | level of expansion |
| *ntype* | Long | Input | Name type (i.e., 7-preferred name, 6-preferred abbreviation, etc.) |
| *szBuffer* | char \* | output | pointer to null terminated string containing the cross expansion string generated |
| *nszBuffer* | long | input | maximum data size szBuffer can hold (including the null terminator). |
| *Aid* | long \* | output | an array of long integer containing the list of germplasm id |
| *naID* | long | input | maximum size of the idList |

**Returns**

GMS\_ERROR, GMS\_SUCCESS, GMS\_MORE\_DATA ,GMS\_NO\_DATA

**Comments**

Progenitor’s preferred name is the default name used when *ntype* is zero. A *generation* is taken to be a generative process involving only derivative progenitors or a sequence of generative processes with the last one involving only derivative progenitors. Thus a series of back crosses is one generation (back to the original single cross) and a triple cross or double cross generation involves two generative steps.

If the input parameter *ntype* is specified (>0) the expansion is built from progenitors having names of the specified type or preferred names of progenitors *level* generations back in the pedigree, whichever occurs first.

Generative progenitors with one parent (mutations) are represented as <parent> /. And ones with more than two parents as <parent1>/[<parent2>,<parent3>, …]. GMS\_crossExpansion will not expand parents other than the first one.

The routine returns GMS\_MORE\_DATA if either the szBuffer or idList is not large enough to hold the data. If the routine encounters a missing group ID in any source expansion of that branch it terminates with the name of that source. If it encounters a missing progenitor ID in any cross before *level* generations it places a query ’?’ in the cross expansion at that point and terminates expansion of that branch.

**GMS\_listNames**

GMS\_listNames searches the database for all names of germplasm *gid* and returns them in a null delimited test string.

**Syntax**

long **GMS\_listNames**(gid,szBuffer,nszBuffer)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **type** | **Use** | **Description** |
| *gid* | long | input | germplasm id |
| *szBuffer* | char \* | output | pointer to null terminated string containing the list of Names |
| *nszBuffer* | long | input | maximum data size szBuffer can hold (including the null terminator). |

**Returns**

GMS\_ERROR, GMS\_SUCCESS, GMS\_MORE\_DATA ,GMS\_NO\_DATA

**Comments**

Names in szBuffer are separated by null characters and the whole list is terminated with two null characters. The function only searches the central database when *gid* is positive, but it always searches the local database. The preferred name (NSTAT = 1) is placed first in the list.

**GMS\_computeGenerationNo**

GMS\_computeGenerationNo traces pedigrees through derivative and management generations and returns statistics indicating the number of each type of generation advance - derivative or management, and the last generation at which seed was harvested from a single plant or at which chromosome doubling was carried. GMS\_computeGenerationNo also returns the type of method used in the last generation and/or the type of generative method most recently used. If the last generative method is a backcross, GMS\_computeGenerationNo returns the number of crosses to the recurrent parent.

**Syntax:**

long **GMS\_computeGenerationNo**(gid,LSPS,NGEN,MAN,NBC,LDM,MCR)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *gid* | long | input | Germplasm ID from which the generation number will be computed |
| *LSPS* | long \* | output | Number of derivative generations from the cross to the last single plant selection or single seed descent (zero for F1s) |
| *NGEN* | long \* | output | Total number of generations from the cross (1 for F1s) |
| *MAN* | long\* | output | Number of management generations |
| *NBC* | long \* | output | Backcross dosage (zero for other cross types) |
| *LDM* | Long \* | output | ICIS method code for the last derivative method (zero for F1s) |
| *MCR* | long \* | output | ICIS method code for the last generative method |

**Returns**

GMS\_SUCCESS ,GMS\_ERROR or GMS\_NO\_DATA

**Comments**

NGEN and LSPS return with zero if there is an unknown source (GPID2=0) in any generation after the cross. This should usually be interpreted as complete inbreeding for self pollinating crops and as F1s for open pollinating species. If there are no unknown sources NGEN returns the number of derivative generations from the cross to the current generation, counting 1 for the cross and ignoring any management generations. If a double haploid method is used (method numbers 202 or 502 in Annex 2.2.b then LSPS returns minus the number of the derivative generation after the cross when this occurred otherwise LSPS returns the derivative generation number after the cross where the last single plant selection (method numbers 205, 504 in Annex 2.2.b) or single seed descent occurred (method numbers 208, 505 in Annex 2.2.b) or zero of none of these methods occur.

MAN returns the number of generations where management methods (MTYPE=’MAN’ in the METHODS Table) have been used between the group source and the current generation. NBC returns the number of crosses to a recurrent parent if the crossing method is backcross (method numbers 107 or 4 in Annex 2.2.b) otherwise NBC=0.

**Code Example:**

I4=GMS\_computeGenerationNo(GID,LSPS,NGEN,MAN,NBC,LDM,MCR)

WRITE(0,130) LSPS,NGEN,MAN,NBC,LDM,MCR

130 FORMAT(6I8)

For IR 80000-B-B-S-B-B-B-B-B where B stands for bulk and S stands for single plant selection, this gives:

3 9 0 0 204 101

Notice that IR 80000 labels the F1, IR 80000-B labels the plot of F2s, IR 80000-B-B labels the plot of F3s and IR 80000-B-B-S represents the F4 seed from one F3 plant and labels the F4 row, ie IR 80000-B-B-S is an F3 derived F4.

**GMS\_getDerivativeNeighbor**

GMS\_getDerivativeNeighbor retrieves all germplasm in the derivative neighborhood of the given GID, m-step backward and n-step forward

**Syntax**

long **GMS\_getDerivativeNeighbor** (*gid, MStep,NStep,idList,idLstSz, blnDH*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| gid | long | input | The GID of the target germplasm |
| MStep | long | input | Number of step backward within the generation history from the level of the target germplasm |
| NStep | long | input | Number of step forward within the generation history from the level of the target germplasm |
| idList | long \* | output | Array that will contain the germplasm within the specified level of derivative neighborhood |
| idLstSz | long \* | input/output | As input, it is the maximum length of idList and as output, it is the number of germplasms retrieved if lower than the maximum length |
| blnDH | long | input | 1 if Double Haploid germplasm is to be excluded as part of derivative neighborhood |

Note:

* Double Haploid option is not yet working very well
* Lines with derivative or management methods (GNPGS < 0) are considered.
* The algorithm is a recursion which is called n+m times

Algorithm:

1. Determine the root of the required neighborhood which is either the germplasm m backward step within the generation history of the specified target GID or its group germplasm if m is greater than generation level of the target GID
2. Starting from the root, get the immediate derived lines
3. For each retrieved line, get the succeeding derived lines
4. Repeat the 3rd step (n+m-2) times

**GMS\_getMgmntNeighbor**

GMS\_getMgmntNeighbor retrieves the management neighbors of the given management group germplasm

**Syntax**

long **GMS\_getMgmntNeighbor** (*mgid, data, name, szName, nszName, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| mgid | long | input | The GID of the root of the management group |
| data | GMS\_Germplasm\* | output | The germplasm record of the management neighbor |
| name | GMS\_NameData\* | output | The name record of the management neighbor |
| szName | Char\* | output | Array of characters for the preferred name of the management neighbor |
| nszName | long | input | length of the szName |
| fopt | long | input | 0 if getting the first record, 1 for succeeding record |

**LIST DLL FUNCTIONS**

The GMS\_openDatabase function mentioned in Section 3.1 should be called first before calling any functions mentioned in this section. The different data structures used by the functions are discussed in section 2.

**FUNCTIONS ASSOCIATED TO LISTNMS TABLE**

**GMS\_getListName**

GMS\_getListName retrieves a specific record from LISTNMS table for a particular value of *listid* field of GMS\_ListName structure. To retrieve all records in the table initialize *listid* to zero, *fOpt* to FIND\_FIRST for the first call and FIND\_NEXT for the succeeding calls.

**Syntax**

long **GMS\_getListName**(*listName*, *fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listName* | GMS\_ListName \* | input/output | Address of a GMS\_ListName structure to contain the list name for the retrieved record |
| *FOpt* | long | output | either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_NO\_DATA, GMS\_ERROR

**GMS\_findListName**

GMS\_findListName retrieves records from LISTNMS table where the list name matches the given string szLName. Set *fOpt* to FIND\_FIRST for the first call and FIND\_NEXT for the succeeding calls.

**Syntax**

long **GMS\_getListName**(*szLName*,*listName*, *fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *szLName* | String | input | Address of NULL terminated string containing the name to search |
| *listName* | GMS\_ListName \* | input/output | Address of a GMS\_ListName structure to contain the list name for the retrieved record |
| *FOpt* | long | output | either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_NO\_DATA, GMS\_ERROR

**GMS\_setListName**

GMS\_setListName changes the value of all the fields in the local LISTNMS table except the LISTID field for a given *listid*.

**Syntax**

long **GMS\_setListName**(*listName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listName* | GMS\_ListName \* | input | Address of a GMS\_ListName structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_addListName**

GMS\_addListName adds one record to the LISTNMS table assigning value automatically to LISTID field.

**Syntax**

long **GMS\_addListName**(*listName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listName* | GMS\_ListName \* | input | Address of a GMS\_ListName structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_deleteListName**

GMS\_deleteListName deletes one record from the LISTNMS table for a given *listid*.

**Syntax**

long GMS\_deleteListName(*listid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listid* | Long | input | value of LISTID of which LISTNAME record to delete |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_getListByStatus**

GMS\_getListByStatus retrieves records from LISTNMS table for the given list status in the GMS\_ListName structure. Set *fOpt* to FIND\_FIRST for the first call and FIND\_NEXT for the succeeding calls.

**Syntax**

long **GMS\_getListByStatus**(*listName*, *fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listName* | GMS\_ListName \* | input/output | Address of a GMS\_ListName structure to contain the list name for the retrieved record |
| *FOpt* | Long | output | either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_NO\_DATA, GMS\_ERROR

**FUNCTIONS ASSOCIATED WITH LISTDATA TABLE**

**GMS\_getListData**

GMS\_getListData retrieves records from the LISTDATA table for given *listid* and *gid* or *listid* and *entryid* values specified in the LISTID, GID and ENTRYID fields of the GMS\_ListData structure.A valid *listid* value must be specified, but the *gid* and *entryid* may be zero inwhich case all entries for the specified list will be retrieved. If *gid* is non zero, *entryid* must be zero and entries with specified *listid* and *gid* will be retrieved. If *entryid* is non zero, *gid* must be zero and the entry with the specified *listid* and *entryid* will be retrieved. *fOpt* should be set equal to FIND\_FIRST on the first call, and FIND\_NEXT in the succeeding calls. (If *entryid* is specified, *fOpt* should be FIND\_FIRST since there can only be one result record.) Search parameters are set when *fOpt* is FIND\_FIRST, and input values in the ListData argument are ignored when *fOpt* is FIND\_NEXT.

**Syntax**

long GMS\_getListData(*listData*, *fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listData* | GMS\_ListData \* | input/output | Address of a GMS\_ListData structure |
| *fOpt* | long | output | either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_findListData**

GMS\_findListData retrieves all records from the LISTDATA table for given *gid* value specified in the GID field of the GMS\_ListData structure. *fOpt* should be set equal to FIND\_FIRST on the first call, and FIND\_NEXT in the succeeding calls. Search parameters are set when *fOpt* is FIND\_FIRST, and input values in the ListData argument are ignored when *fOpt* is FIND\_NEXT.

**Syntax**

long GMS\_getListData(*listData*, *fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listData* | GMS\_ListData \* | input/output | Address of a GMS\_ListData structure |
| *fOpt* | long | output | either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_setListData**

GMS\_setListData changes the value of fields in the local LISTDATA table for given *listid* and *lrecid* fields of GMS\_ListData structure. (Note the LISTID and LRECID fields cannot be changed.)

**Syntax**

long **GMS\_setListData**(*listData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listData* | GMS\_ListData \* | input | Address of a GMS\_ListData structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_addListData**

GMS\_addListData adds one record to the LISTDATA table assigning value automatically to ENTRYID field.

**Syntax**

long **GMS\_addListData**(*listData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listData* | GMS\_ListData \* | input | Address of a GMS\_ListData structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**GMS\_deleteListData**

GMS\_deleteListData deletes records from the LISTDATA table for a given *listid* or, *listid* and *lrecid*. If *lrecid* is zero the function will delete all records where *listid*=LISTID, otherwise delete record where *listid*=LISTID and *entryid*=ENTRYID.

**Syntax**

long **GMS\_deleteListData**(*listid*, *lrecid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *listid* | Long | input | value of LISTID of which LISTDATA record to delete |
| *lrecid* | Long | input | value of LRECID, if LRECID is not zero, with the condition above applied of which LISTDATA record to delete |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**USER DEFINED FIELDS AND CONSTANTS FUNCTIONS**

**GMS\_getUDField**

GMS\_getUDField will retrieve details of any user-defined field from the UDFLDS table. The first argument of GMS\_getUDField is a pointer to a GMS\_UDField structure. If the FLDNO is not zero when the function is called then it returns the details for the corresponding user defined field. If it is zero then the function returns details for the first user defined field if *fOpt* = FIND\_FIRST or for the next available user defined field if *fOpt* = FIND\_NEXT. Any call which fails to get user defined field information returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getUDField**(UDField, szFDesc, nszFDesc, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| UDField | GMS\_UDField\* | input/ output | Address of a GMS\_UDField structure |
| szFDesc | char \* | output | Address of a string variable to contain the field description as a NULL terminated string |
| nszFDesc | Long | input | Maximum size of *szFDesc* |
| fOpt | Long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The principle of using GMS\_getUDField is almost the same as GMS\_getAttribute. If the function is called and there was a problem accessing the database, the function will return GMS\_ERROR.

**GMS\_getUDField2**

GMS\_getUDField2 will retrieve the allowable constant values for a column (FTYPE) of an ICIS table (FTABLE). The first argument is a pointer to a GMS\_UDField structure where FTABLE and FTYPE are given. If fopt is FIND\_FIRST (0), the function retrieves the first matched record. If fopt is FIND\_NEXT (1), details about the succeeding matched record is retreived .Any call which fails to get user defined field information returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getUDField2**(UDField, szFDesc, nszFDesc, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| UDField | GMS\_UDField\* | input/ output | Address of a GMS\_UDField structure |
| szFDesc | char \* | output | Address of a string variable to contain the field description as a NULL terminated string |
| nszFDesc | Long | input | Maximum size of *szFDesc* |
| fOpt | Long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GMS\_findUDField**

GMS\_findUDField searches for UDFLDS record with name matching the string FNAME. The first argument is a pointer to a GMS\_UDField structure where FNAME is given. If fopt is FIND\_FIRST (0), the function retrieves the first matched record. If fopt is FIND\_NEXT (1), details about the succeeding matched record is retreived .Any call which fails to get user defined field information returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_findUDField**(UDField, szFDesc, nszFDesc, fOpt)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| UDField | GMS\_UDField\* | input/ output | Address of a GMS\_UDField structure |
| szFDesc | char \* | output | Address of a string variable to contain the field description as a NULL terminated string |
| nszFDesc | Long | input | Maximum size of *szFDesc* |
| fOpt | Long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**DMS DLL FUNCTIONS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

**INTRODUCTION**

The DMS\_openDatabase function mentioned should be called first before calling any functions mentioned in this section. The different data structures used by the functions are discussed in section IV.I.2.

**ADD FUNCTIONS**

**DMS\_addStudy**

DMS\_addStudy adds new study record in the DMS database according to the information in the STUDY argument. STUDYID is automatically assigned by the function. The name of the study should be unique. If a study with similar name already exists in the database, the function returns DMS\_EXIST (-5).

**Syntax**

long **DMS\_addStudy**(*STUDY, szObjective*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| STUDY | DMS\_STUDY | Input | Address of the DMS\_STUDY structure that contains the information to be added to the database. |
| szObjective | Char\* | Input | The pointer to a null terminated string containing the objective of the Study |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_ERROR(-1)

**DMS\_addFactor**

DMS\_addFactor adds new factor of a given study (STUDYID) according to the information in the FACTOR argument. FACTORID is automatically assigned by the function. Factor name should be unique within a study. Hence, if a factor with similar name exists in the given study, the function returns DMS\_EXIST (-5). If the STUDYID does not exist in the database, DMS\_INVALID\_ID (-9) is returned.

**Syntax**

long **DMS\_addFactor**(*FACTOR)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| FACTOR | DMS\_FACTOR | Input | Address of the DMS\_FACTOR structure that contains the information to be added to the database . |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_INVALID\_ID(-9),DMS\_ERROR(-1)

**DMS\_addVariate**

DMS\_addVariate adds new variate of a given study (STUDYID) according to the information in the VARIATE argument. VARIATID is automatically assigned by the function. Variate name should be unique within the study. Hence, if a variate with similar name exists in the given study, the function returns DMS\_EXIST (-5). If the STUDYID does not exist in the database, DMS\_INVALID\_ID (-9) is returned.

**Syntax**

long **DMS\_addVariate**(*VARIATE*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| VARIATE | DMS\_VARIATE | Input | Address of the DMS\_VARIATE structure that contains the information to be added to the database . |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_INVALID\_ID(-9),DMS\_ERROR(-1)

**DMS\_addLevelN**

**DMS\_addLevelN** adds new level of a particular factor (FACTORID) having numeric type according to the information in the LEVELN argument. LEVELNO is assigned by the function and it is unique within a factor. If there is a level of similar value for that factor, DMS\_EXIST is returned by the function. If the FACTORID does not exist in the database, DMS\_INVALID\_ID (-9) is returned.

**Syntax**

long **DMS\_addLevelN**(*LEVELN*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LEVELN | DMS\_LEVELN | Input | Address of the DMS\_LEVELN structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_INVALID\_ID(-9),DMS\_ERROR(-1)

**DMS\_addLevelC**

DMS\_addLevelC adds new level of a particular factor (FACTORID) having character type according to the information in the LEVELC argument. The LEVELNO is assigned by the function and it is unique within a factor. If there is a level of similar value for that factor, DMS\_EXIST is returned by the function. If the FACTORID does not exist in the database, DMS\_INVALID\_ID (-9) is returned.

**Syntax**

long **DMS\_addLevelC***(LEVELC*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LEVELC | DMS\_LEVELC | Input | Address of the DMS\_LEVELC structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_INVALID\_ID(-9),DMS\_ERROR(-1)

**DMS\_addDataN**

DMS\_addDataN adds new data value for a particular variate of numeric type according to the information in the DATAN argument. If either OUNITID or VARIATID does not exist in the database, the function returned DMS\_INVALID\_ID (-9).

**Syntax**

long **DMS\_addDataN**(*DATAN*);

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| Argument | Type | Use | Description |
| DATAN | DMS\_DATAN | Input | Address of the DMS\_DATAN structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_INVALID\_ID(-9), DMS\_ERROR(-1)

**DMS\_addDataC**

DMS\_addDataC adds new data value for a particular variate of character type according to the information in the DATAC argument. If either OUNITID or VARIATID does not exist in the database, the function returned DMS\_INVALID\_ID (-9).

**Syntax**

long **DMS\_addDataC**(*DATAC*);

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| DATAC | DMS\_DATAC | Input | Address of the DMS\_DATAC structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_INVALID\_ID(-9), DMS\_ERROR(-1)

**DMS\_addObsunit**

DMS\_addObsUnit creates new observation unit for a given study. The information required by the function is the STUDYID. A new observation unit (OUNITID) is assigned by the function. If STUDYID is non-existing, DMS\_EXIST (-5) is returned.

**Syntax**

long **DMS\_addObsunit**(*OBS*);

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| OBS | DMS\_OBSUNIT | Input | Address of the DMS\_OBSUNIT structure that contains the information to be added to the database . |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_ERROR(-1)

**DMS\_addOindex**

DMS\_addOindex adds new factor level defining an observation unit. OUNITID, FACTORID and LEVELNO should all be existing in the database or else the function returns DMS\_INVALID\_ID (-9).

**Syntax**

long **DMS\_addOindex**(*OINDEX*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| OINDEX | DMS\_OINDEX | Input | Address of the DMS\_OINDEX structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_INVALID\_ID(-9), DMS\_ERROR(-1)

**0 DMS\_addAttr**

DMS\_addAttr adds new attribute associated to either the STUDY,FACTOR or VARIATE table.to the DMSATTR table.

**Syntax**

long **DMS\_addAttr**(Attr)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Attr | DMS\_ATTR | Input | Address of the DMS\_DMSATTR structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_addTrait**

DMS\_addTrait adds new trait according to the information in the TRAIT argument with trait name being unique. TRAITID is automatically assigned by the function.

**Syntax**

long **DMS\_addTrait**(*Trait,szObj*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Trait | DMS\_TRAIT | Input | Address of the DMS\_TRAIT structure that contains the information to be added to the database. |
| szObj | Char\* | Input | Address of a null terminated string that contains the objective of the Study. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_addScale**

DMS\_addScale adds new scale for a triat according to the information in the SCALE argument with scale name being unique within the trait. SCALEID is automatically assigned by the function.

**Syntax**

long **DMS\_addScale***(Scale)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Scale | DMS\_SCALE | Input | Address of the DMS\_SCALE structure that contains the information to be added to the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_addTmethod**

DMS\_addTmethod adds new method for a trait according to the information in the TMETH argument with method name being unique within the trait. TMETHID is automatically assigned by the function.

Syntax

long **DMS\_addTmethod***(Tmeth,szDesc)*

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Tmeth | DMS\_TMETHOD | Input | Address of the DMS\_TMETHOD structure that contains the information to be added to the database. |
| szDesc | Char\* | Input | Address of a null terminated string that contains the description of the method. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_addEffect**

DMS\_addEffect adds new representation of an effect by appending the list of factors that defines the representation to the EFFECT table.

Syntax

long **DMS\_addEffect** *(lstfactor, nfactor, represno, effectid)*

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| lstfactor | long\* | Input | Address of a list of factors that defines the representation of an effect |
| nfactor | long | Input | The number of factors in the list. |
| represno | Long\* | Output | The assigned number of the representation |
| effectid | Char\* | Input | Address of a null terminated string that contains the description of the method. |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_INVALID\_ID(-9),DMS\_ERROR(-1)

**DMS\_addVEffect**

DMS\_addVEffect adds a variate of an effect representation to the VEFFECT table.

**Syntax**

long **DMS\_addVEffect** *(veffect))*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| veffect | DMS\_VEFFECT | Input | Address of the DMS\_VEFFECT structure to contain the representation number and the variate id. |

**Returns**

DMS\_SUCCESS(1), DMS\_EXIST(-5), DMS\_ERROR(-1)

**DMS FIND FUNCTIONS**

The find functions search the DMS database based on a give name or value.

**DMS\_findStudy**

**DMS\_findStudy** searches for study with name matching the string *szStudy*. If fopt is FIND\_FIRST (0), the function stores the study details of the first matched record. If fopt is FIND\_NEXT (1), details about the succeeding matched record is retreived .

**Syntax**

Long **DMS\_findStudy**(*szStudy, STUDY, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| szStudy | Char\* | Input | Address of NULL terminated string containing the string to be matched |
| STUDY | DMS\_STUDY | Output | Address of the DMS\_STUDY structure to contain the study data of the retrieved record. |
| fopt | Long | Input | Either FIND\_FIRST(0) or FIND\_NEXT(1) |

**Returns**

DMS\_SUCCESS(1), DMS\_NO\_DATA(0), DMS\_ERROR(-1)

**DMS\_findFactor**

**DMS\_findFactor** searches for factor with name matching *szName*. If STUDYID is zero, the function looks for all factors having the matched name; otherwise, the function considers factor names within the given study. If fopt is FIND\_FIRST (0), the function stores the factor details of the first matched record. If fopt is FIND\_NEXT, details about the succeeding matched record is stored..

**Syntax**

long **DMS\_findFactor**(*szName, FACTOR, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| szName | Char\* | Input | Address of NULL terminated string containing the string to be matched |
| FACTOR | DMS\_FACTOR | Output | Address of the DMS\_FACTOR structure to contain the data of the retrieved record. |
| fopt | Long | Input | Either FIND\_FIRST(0) or FIND\_NEXT(1) |

**Returns**

DMS\_SUCCESS(1), DMS\_NO\_DATA(0), DMS\_ERROR(-1)

**DMS\_findVariate**

DMS\_findVariatesearches for variate with name matching *szName*. If STUDYID is zero, the function looks for all variates having the matched name; otherwise, the function considers variates within the given study. If fopt is FIND\_FIRST (0), the function stores the variate details of the first matched record. If fopt is FIND\_NEXT, details about the succeeding matched record is stored..

**Syntax**

long **DMS\_findVariate**(*szName, VARIATE, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| szName | Char\* | Input | Address of NULL terminated string containing the string to be matched |
| VARIATE | DMS\_VARIATE | Output | Address of the DMS\_VARIATE structure to contain the variate data of the retrieved record. |
| fopt | Long | Input | Either FIND\_FIRST(0) or FIND\_NEXT(1) |

**Returns**

DMS\_SUCCESS(1), DMS\_NO\_DATA(0), DMS\_ERROR(-1)

**DMS\_findLevelN**

DMS\_findLevelN returns the LEVELNO of numeric value equal to LVALUE under the factor identified by LABELID.

**Syntax**

long **DMS\_findLevelN**(*LevN*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevN | DMS\_LEVELN  FACTORID LABELID LEVELNO LEVEL\_VALUE | Output  Output Input Output Input | Address of the DMS\_LEVELN structure to contain the given factor identification number and the level value, and the retrieved LEVELNO. |

**DMS\_findLevelC**

DMS\_findLevelC returns the LEVELNO of character value equal to LVALUE under the factor identified by LABELID.

**Syntax**

long **DMS\_findLevelC**(*LevC*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevC | DMS\_LEVELC  LABELID FACTORID LEVELNO LEVEL\_VALUE | Output  Output Input Output Input | Address of the DMS\_LEVELC structure to contain the given factor identification and the character level value, and the retrieved LEVELNO . |

**DMS\_findLabelC**

**DMS\_findLabelC** searches for character label value equal to LVALUE identified by LABELID, FACTORID and LEVELNO.

**Syntax**

long **DMS\_findLabelC**(*LevC*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevC | DMS\_LEVELC  FACTORID LEVELNO LEVEL\_VALUE | Output  Input Output Input | Address of the DMS\_LEVELC structure to contain the given label, factor and level number identification and the character label value. |

**DMS\_findLabelN**

**DMS\_findLabelN** searches for numeric label value equal to LVALUE identified by LABELID, FACTORID and LEVELNO.

**Syntax**

long **DMS\_findLabelN**(*LevN*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevN | DMS\_LEVELN  FACTORID LABELID LEVELNO LEVEL\_VALUE | Output  Input Input Output Input | Address of the DMS\_LEVELN structure to contain the given factor identification number and the level value, and the retrieved LEVELNO. |

**DMS\_findStOunit**

**DMS\_findStOunit** finds the observation unit that contains the global factors particularly the study factor. The input argument StuChar should have the trait id of the Study factor and name of the study stored in CVALUE element.

Syntax

long **DMS\_findStOunit**(*StuChar, fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| StuChar | DMS\_CharVALUE   Ounitid traitid scaleid tmethid cvalue | Output/ Input  Output Input   Input | Address of the DMS\_CharVALUE structure to contain the study trait id and the study name, and the retrieved OUNITID . |
| fopt | Long | Input | Either FIND\_FIRST(0) or FIND\_NEXT(1) |

**DMS\_findStEffect**

**DMS\_findStEffect** finds the effect representation that is defined by the global factors particularly the study factor. The input argument StuChar should have be the TRAITID of the Study factor and a particular study name stored in CVALUE.

**Syntax**

long **DMS\_findStEffect**( *studyid, represno*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Studyid | Long | Input | The id of the given study. |
| Represno | Long\* | Output | The number of the representation defined by the global factors. |

**DMS GENERAL QUERY FUNCTIONS**

**DMS\_searchLevelN**

DMS\_searchLevelN searches for the numeric level values of any factor belonging to a specific trait. If *scaleid* is non-zero, the function will only extract level data expressed by the given scale. If t*methid* is non-zero, then level data measured through the given method will be retrieved. If *scaleid* and *tmethid* are both zero, all level values of the trait will be extracted.

Syntax

Long **DMS\_searchLevelN**( *traitid, scaleid, tmethid, ounitid, flevelN, fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| traitid | Long | Input | The identification no. of the trait you want to get. |
| scaleid | Long | Input | The identification no. of the trait’s scale. |
| tmethid | Long | Input | The identification no. of the trait’s method. |
| ounitid | Long | Output | The observation unit having value for the specified trait. |
| flevelN | double | Output | The level value of the trait of numeric type. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR (-1), DMS\_NO\_DATA (0)

**DMS\_searchDataN**

**DMS\_searchDataN** searches for all numeric values of any variable belonging to a specified trait. If *scaleid* is non-zero, the function will extract data expressed by the given scale. If *tmethid* is non-zero, then data observed by the given method will be retrieved. If *scaleid* and *tmethid* are both zero, all data values of the trait will be extracted.

Syntax

Long **DMS\_searchDataN**( *traitid, scaleid, tmethid, ounitid, dvalueN, fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| traitid | Long | Input | The identification no. of the trait you want to get . |
| scaleid | Long | Input | The identification no. of the trait’s scale. |
| tmethid | Long | Input | The identification no. of the trait’s method . |
| Ountid | Long | Output | The observation unit having value for the specified trait. |
| dvalueN | Double | Output | The data value of the trait of numeric type. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR (-1), DMS\_NO\_DATA (0)

**DMS\_searchLevelC**

**DMS\_searchLevelC** searches for all character level values of any factor belonging to a specific trait. If *scaleid* is non-zero, the function will extract level data expressed by the given scale. If *tmethid* is non-zero, then factor level applied through the given method will be retrieved. If *scaleid* and *tmethid* are both zero, all level values of the trait will be extracted.

Syntax

Long **DMS\_searchLevelC**( *traitid, scaleid, tmethid, ounitid , flevelC, fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| tratid | Long | Input | The identification no. of the trait you want to get. |
| scaleid | Long | Input | The identification no. of the trait’s scale. |
| tmethid | Long | Input | The identification no. of the trait’s method. |
| Ounit | Long | Output | The observation unit having value for the specified trait. |
| flevelC | Char[255] | Output | The level value of the trait of character type. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR (-1), DMS\_NO\_DATA (0)

**DMS\_searchDataC**

**DMS\_searchDataN** searches for all character values of any variable belonging to a specified trait. If *scaleid* is non-zero, then the function will only extract data expressed by the given scale. If *tmethid* is non-zero, then data values which are observed through the given method will be retrieved. If *scaleid* and *tmethid* are both zero, all data values of the trait will be extracted.

Syntax

Long **DMS\_searchDataC**(*traitid, scaleid, tmethid, ounitid,* *dvalueC, fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Traitid | Long | Input | The identification no. of the trait you want to get. |
| Scaleid | Long | Input | The identification no. of the trait’s scale. |
| Tmethid | Long | Input | The identification no. of the trait’s method. |
| Ounitid | Long | Output | The observation unit having value for the specified trait. |
| dvalueC | Char[255] | Output | The data value of the trait of character type. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR (-1), DMS\_NO\_DATA (0)

**QUERY THROUGH OBSERVATION UNIT**

**DMS\_defineOunitDN**

DMS\_defineOunitDN retrieves the trait and the character data value describing a given observation unit. The trait value is searched from the DATA\_N table. The function outputs the trait, the scale for expressing it, the method for measuring it and its value.

**Syntax**

Long **DMS\_defineOunitDN**(*NumValue,fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| NumValue | DMS\_NumVALUE   .ounitid  .traitid .scaleid .tmethid .nvalue | Input/ Output  Input  Output | The observation unit being described.  The numeric value of the factor level which is one of those that defines the observation unit, together with its traitid, scaleid, and tmethid |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_defineOunitDC**

DMS\_defineOunitDN retrieves the trait and the character data value describing a given observation unit. The trait value is searched from the DATA\_C table. The function outputs the trait, the scale for expressing it, the method for measuring it and its value.

**Syntax**

Long **DMS\_defineOunitDC**(*CharValue,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| CharValue | DMS\_CharVALUE   .ounitid  .traitid .scaleid .tmethid .cvalue | Input/ Output  Input  Output | The observation unit being described.  The numeric value of the factor level which is one of those that defines the observation unit, together with its traitid, scaleid, and tmethid |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_defineOunitLN**

DMS\_defineOunitLN retrieves the trait and the numeric level value describing the specified observation unit. . The function outputs the factor trait, the scale for expressing it, the method for measuring it and its value.

**Syntax**

Long **DMS\_defineOunitLN**(*NumValue,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| NumValue | DMS\_NumVALUE   .ounitid  .traitid .scaleid .tmethid .nvalue | Input/ Output  Input  Output | The observation unit being described.  The numeric value of the factor level which is one of those that defines the observation unit, together with its traitid, scaleid, and tmethid |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_defineOunitLC**

DMS\_defineOunitLC retrieves the trait and the character level value describing the given observation unit. . The function outputs the factor trait, the scale for expressing it, the method for measuring it and its value.

**Syntax**

Long **DMS\_defineOunitLC**(*CharValue,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| CharValue | DMS\_CharVALUE   .ounitid  .traitid .scaleid .tmethid .cvalue | Input/ Output  Input  Output | The observation unit being described.  The numeric value of the factor level which is one of those that defines the observation unit, together with its traitid, scaleid, and tmethid |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_extractOunitDN**

DMS\_extractOunitDN extracts the observation unit from a given *studyid* having trait value associated to the specified *dblValue.* The trait value can be equal, greater than, less than, greater than or equal or less than or equal to the *dblValue* depending on the specified *operator*. The trait is represented by the traitid, expressed by *scaleid* and measured by *tmethid*. The trait value is searched from the DATA\_N table. The function returns DMS\_ERROR if no observation unit found.

**Syntax**

Long **DMS\_extractOunitDN** (*studyid,dblValue,NumValue,operator,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | Long | Input |  |
| dblValue | Double | Input | The value to compare. |
| NumValue | DMS\_NumVALUE  .ounitid  .traitid  .scaleid   .tmethid   .nvalue | Input/ Output Output  Input  Input   Input   Output | The identification no. of the trait you want to get.  The observation unit where the trait is observed.  The identification no. of the trait.  The identification no. of the scale used in expressing the trait.  The identification no. of the method used in measuring the trait.  The retrieved numeric data value. |
| operator | Long | Input | Can be any of the following: DMS\_EQ(0),DMS\_GT(1),DMS\_LT(2), DMS\_GTE(3),DMS\_LTE(4) |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_extractOunitDC**

DMS\_extractOunitDC extracts the observation unit from a given *studyid* having trait value equal to the specified *cvalue.* The trait is represented by the *traitid*, expressed by *scaleid* and measured by *tmethid*. The trait value is searched from the DATA\_C table.. The function returns DMS\_ERROR if no observation unit found.

**Syntax**

Long **DMS\_extractOunitDC** *(studyid,szValue,CharValue,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| CharValue | DMS\_CharVALUE   .ounitid   .traitid  .scaleid   .tmethid   .cvalue | Input/ Output  Output   Input  Input   Input   Input | The identification no. of the trait you want to get.  The observation unit where the trait is observed.  The identification no. of the trait.  The identification no. of the scale used in expressing the trait.  The identification no. of the method used in measuring the trait.  The character data value to be compared. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_extractOunitLN**

DMS\_extractOunitLN extracts the observation unit from a given *studyid* having trait value associated to the specified *dblValue.* The trait value can be equal, greater than, less than, greater than and equal or less than and equal to the *dblValue* depending on the specified *operator*. The trait is represented by the *traitid*, expressed by *scaleid* and measured by *tmethid*. The trait value is searched from the LEVEL\_N table... The function returns DMS\_ERROR if no observation unit found.

**Syntax**

Long **DMS\_extractOunitLN**(*studyid,dblValue,NumValue,operator,fopt*)

Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | Long | Input |  |
| dblValue | Double | Input | The value to compare. |
| NumValue | DMS\_NumVALUE   .ounitid   .traitid  .scaleid   .tmethid   .nvalue | Input/ Output  Output   Input  Input   Input   Output | The identification no. of the trait you want to get.  The observation unit where the trait is observed.  The identification no. of the trait.  The identification no. of the scale used in expressing the trait.  The identification no. of the method used in measuring the trait.  The numeric data value to be compared. |
| operator | Long | Input | Can be any of the following: DMS\_EQ(0),DMS\_GT(1),DMS\_LT(2), DMS\_GTE(3),DMS\_LTE(4) |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_extractOunitLC**

DMS\_extractOunitLC extracts the observation unit from a given *studyid* having trait value equal to the specified *szValue.* The trait is represented by the *traitid*, expressed by *scaleid* and measured by *tmethid*. The trait value is searched from the LEVEL\_C table.. The function returns DMS\_ERROR if no observation unit found.

**Syntax**

Long **DMS\_extractOunitLC** *(studyid,cvalue,CharValue,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| CharValue | DMS\_CharVALUE   .ounitid   .traitid  .scaleid   .tmethid   .cvalue | Input/ Output  Output   Input  Input   Input   Input | The identification no. of the trait you want to get.  The observation unit where the trait is observed.  The identification no. of the trait.  The identification no. of the scale used in expressing the trait.  The identification no. of the method used in measuring the trait.  The character data value to be compared. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_describeOunit**

DMS\_describeOunit returns an array of factor ids and level numbers that defines a specific observation unit.

**Syntax**

Long **DMS\_describeOunit** (*ounitid, arrLev, nArr, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ounitid | Long | Input | The number of the observation unit. |
| arrLev | \*source | Output | The address of the first element of the array that will contain the factor ids and level numbers |
| nArr | Long | Input/ Output | As input, the maximum allowable number of element of the array and as output the number of retrieved factors. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_describeEffectOunit**

DMS\_describeEffectOunit returns an array of factor ids and level numbers that defines a specific observation unit of a given representation.

**Syntax**

Long **DMS\_describeEffectOunit** (*ounitid,represno, arrLev, nArr, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ounitid | Long | Input | The number of the observation unit. |
| represno | Long | Input | The number of the effect representation. |
| arrLev | \*source | Output | The address of the first element of the array that will contain the factor ids and level numbers. |
| nArr | Long | Input/ Output | As input, the maximum allowable number of element of the array and as output the number of retrieved factors. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS ACCESS FUNCTIONS**

These functions searches the DMS based on the key ids of the DMS tables.

**DMS\_getStudy**

DMS\_getStudy retrieves details of a Study. The first argument of the function is a pointer to DMS\_STUDY structure. If the STUDYID element is not zero then the details for the corresponding study is retrieved. If it is zero, the function returns the details for the first study if fOpt = FIND\_FIRST (0) or for the next available study if fOpt = FIND\_NEXT(1). The objective of the study is placed in a separate argument *szObj*. Any call which fails to find a study returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getStudy**(*STUDY, szSObj, cSObj, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| STUDY | DMS\_STUDY | Input/Output | Address of a DMS\_STUDY structure. |
| szSObj | Char \* | Output | Address of a NULL terminated string to place the trait description. |
| cSObj | Long | Input | Maximum size of szTDesc. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getObsunit**

DMS\_getObsunit retrieves the observation unit of a study. The first argument of the fucntion is the study id and the next argument is a pointer to DMS\_OBSUNIT structure. The functions gets the first found observation unit if fOpt = FIND\_FIRST (0) or for the next available observation unit if fOpt = FIND\_NEXT(1). Any call which fails to find an observation unit returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getObsunit**(*studyid,Obs, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | Long | Input | The study where to retrieve the observation unit. |
| Obs | DMS\_OBS | Output | Address of a DMS\_OBSUNIT structure. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getLevelC**

DMS\_getLevelC retrieves the value of a specific level of a character factor. The argument of the function is a pointer to DMS\_LEVELC structure which contains the label id, factor id and the level no that identify the factor value. If the function fails to find a character factor value, it returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getLevelC**(*LevC*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevC | DMS\_LEVELC | Input/Output | The address of the DMS\_LEVELC structure. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getLabelC**

DMS\_getLabelC retrieves the label ids and its associated values for a given level of a factor. The parameters passed to the function are the factor id and level number stored in the DMS\_LEVELC structure, which will also contain the returned label id and value.

**Syntax**

Long **DMS\_getLabelC**(*LevC, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevC | DMS\_LEVELC | Input/Output | The address of the DMS\_LEVELC structure. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getAllLevelC**

DMS\_getAllLevelC retrieves the levels and values of a given character label or factor. The argument of the function is a pointer to DMS\_LEVELC structure, which contains the label id. The level number and the character value of the label are returned to the LEVELNO and LEVEL\_VALUE elements of the data structure. If the function does not find any character factor value, it returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getAllLevelC**(*LevC, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevC | DMS\_LEVELC | Input/Output | The address of the DMS\_LEVELC structure. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getAllLabelC**

DMS\_getAllLabelC retrieves the labels and factor values of a given factorid from the LEVEL\_C table. The argument of the function is a pointer to DMS\_LEVELC structure, which contains the factorid. The value, level no and labeid are returned to the LEVEL\_VALUE, LEVELNO and LABELID elements of the data strcuture. The function returns DMS\_NO\_DATA if no record is found.

**Syntax**

Long **DMS\_getAllLabelC**(*LevC, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevC | DMS\_LEVELC | Input/Output | The address of the DMS\_LEVELC structure. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getLevelN**

DMS\_getLevelC retrieves the value of a numeric factor. The argument of the function is a pointer to DMS\_LEVELN structure which contains the label id, factor id and the level no that identify the factor value. If the function fails to find a numeric factor value, it returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getLevelN**(*LevN*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevN | DMS\_LEVELN | Input/Output | The address of the DMS\_LEVELN structure. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getLabelN**

DMS\_getLabelN retrieves the label ids and its associated values for a given level of a factor. The parameters passed to the function are the factor id and level number stored in the DMS\_LEVELN structure, which will also contain the returned label id and value.

**Syntax**

Long **DMS\_getLabelN**(*LevN, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevN | DMS\_LEVELN | Input/Output | The address of the DMS\_LEVELC structure. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getAllLevelN**

DMS\_getAllLevelN retrieves the levels and values of a given numeric label or factor. The argument of the function is a pointer to DMS\_LEVELN structure, which contains the label id. The level number and the numeric value of the label are returned to the LEVELNO and LEVEL\_VALUE elements of the data structure. If the function fails to find a numeric factor value, it returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getAllLevelN**(*LevN, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevN | DMS\_LEVELN | Input/Output | The address of the DMS\_LEVELN structure. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**0DMS\_getAllLabelN**

DMS\_getAllLabelN retrieves the labels and factor values of a given factorid from the LEVEL\_N table. The argument of the function is a pointer to DMS\_LEVELN structure, which contains the factorid. The value, level no and labeid are returned to the LEVEL\_VALUE, LEVELNO and LABELID elements of the data strcuture. The function returns DMS\_NO\_DATA if no record is found.

**Syntax**

Long **DMS\_getAllLabelN**(*LevN, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LevN | DMS\_LEVELN | Input/Output | The address of the DMS\_LEVELN structure. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getDataC**

DMS\_getDataC retrieves the value of a character data. The argument of the function is a pointer to DMS\_DATAC structure which contains the observation unit id and the variate id that identify the data value. If the function fails to find a character data value, it returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getDataC**(*DatC)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| DatC | DMS\_DATAC | Input/Output | The address of the DMS\_DATAC structure. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getDataN**

DMS\_getDataN retrieves the value of a numeric data. The argument of the function is a pointer to DMS\_DATAN structure which contains the observation unit id and the variate id that identify the data value. If the function fails to find a numeric data value, it returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getDataN**(*DatN)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| DatN | DMS\_DATAN | Input/Output | The address of the DMS\_DATAN structure. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getTrait**

DMS\_getTrait retrieves details of a trait. The first argument of the function is a pointer to DMS\_TRAIT structure. If the TRAITID element is not zero then the details for the corresponding trait is retrieved. If it is zero, the function returns the details for the first trait if fOpt = FIND\_FIRST (0) or for the next available trait if fOpt = FIND\_NEXT(1). The description of the trait is placed in a separate argument *szTDesc*. Any call which fails to find a trait returns DMS\_NO\_DATA.

**Syntax**

Long **DMS\_getTrait**(*TRAIT, szTDesc, cTDesc, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| TRAIT | DMS\_TRAIT | Input/Output | Address of a DMS\_TRAIT structure. |
| szTDesc | Char \* | Output | Address of a NULL terminated string to place the trait description. |
| cTDesc | Long | Input | Maximum size of szTDesc. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getScale**

DMS\_getScale retrieves details of a scale. The first argument of the function is a pointer to DMS\_SCALE structure. If the SCALEID element is not zero then the details for the corresponding scale is retrieved. If it is zero, the function returns the details for the first scale if fOpt = FIND\_FIRST (0) or for the next available scale if fOpt=FIND\_NEXT(1).

**Syntax**

Long **DMS\_getScale**(*SCALE, fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| SCALE | DMS\_SCALE | Input/Output | Address of a DMS\_SCALE structure . |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getScaleCon**

DMS\_getScaleCon retrieves the start value and end value of continuous numeric scale. The first argument of the function is a pointer to DMS\_SCALECON structure. If the SCALEID element is not zero then the range for the corresponding scale is retrieved. If it is zero, the function returns all the ranges of the continuous scales. If fOpt = FIND\_FIRST (0), then first record is retrieved. if fOpt=FIND\_NEXT(1), then the succedding record is retrieved. If no record is found, DMS\_NO\_DATA is return value of the function.

**Syntax**

Long **DMS\_getScaleCon***(ScaleCon; fopt)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ScaleCon | DMS\_SCALECON | Input/Output | Address of a DMS\_SCALECON structure . |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

|  |  |
| --- | --- |
| **5.6.16** | DMS\_getScaleDis |

DMS\_getScaleDis retrieves the allowed value of a discreet scale and its description. The first argument of the function is a pointer to DMS\_SCALEDIS structure. If the SCALEID element is not zero then the allowed values for the given scale are retrieved. If it is zero, the function returns all the values of the discreets scales. If fOpt = FIND\_FIRST (0), then first record is retrieved. if fOpt=FIND\_NEXT(1), then the succeeding record is retrieved. If no record is found, DMS\_NO\_DATA is return value of the function.

**Syntax**

Long **DMS\_getScaleDis***(ScaleDis; fopt)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ScaleDis | DMS\_SCALEDIS | Input/Output | Address of a DMS\_SCALEDIS structure . |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**DMS\_getScaleDis2**

DMS\_getScaleDis2 retrieves the description of the value of a given scaleid. If no record is found, DMS\_NO\_DATA is return value of the function.

**Syntax**

Long **DMS\_getScaleDis2***(ScaleDis; fopt)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ScaleDis | DMS\_SCALEDIS | Input/Output | Address of a DMS\_SCALEDIS structure. The value and scalied are provided as inputs and the description is returned by the function |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**DMS\_getScaleTab**

DMS\_getScaleTab executes the SQL statement stored in the ScaleTab table for the specified scaleid. The result of the query is stored in the given file name and the number of resulted columns is returned by the function.

**Syntax**

Long **DMS\_getScaleTab***(scaleid,fileName,noCol)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| scaleid | long | Input | The identification of the scale whose values are stored in the ICIS tables. . |
| fileName | Char \* | Input | Address of a NULL terminated string that contains the file name where the result of the SQL statement is stored. |
| noCol | Long | Output | The number of columns returned by the SQL statement. |

**DMS\_getTmethod**

DMS\_getTmethod retrieves details of a method. The first argument of the function is a pointer to DMS\_TMETHOD structure. If the TMETHID element is not zero then the details for the corresponding method is retrieved. If it is zero, the function returns the details for the first method if fOpt = FIND\_FIRST (0) or for the next available method if fOpt = FIND\_NEXT(1).

**Syntax**

Long **DMS\_getTmethod**(*TMETHOD,* szTmethDesc,cTmethDescn,*fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| TMETHOD | DMS\_TMETHOD | Input/Output | Address of a DMS\_TMETHOD structure. |
| szTmethDesc | Char \* | Output | Address of a NULL terminated string to place the method description. |
| cTmethDesc | Long | Input | Maximum size of szTmethDesc. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getOunitLevelN**

DMS\_getOunitLevelN retrieves an observation unit of a numeric factor level. The first argument of the function is a pointer to LEVELN structure. Values for the LABELID and FACTORID elements must be provided. The corresponding observation unit for the LABELID and FACTORID is retrieved, along with the LEVELNO and LVALUE values. If fOpt = FIND\_FIRST (0), details of the first level value is retrieved, or the next available level value, of fOpt = FIND\_NEXT(1).

**Syntax**

Long **DMS\_getOunitLevelN**(*LEVELN,ouid,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LEVELN | DMS\_LEVELN | Input/Output | Address of a DMS\_LEVELN structure. |
| ounitid | Long | Output | The observation unit number being defined by the combination of factors and levels. |
| Fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getOunitLevelC**

DMS\_getOunitLevelC retrieves an observation unit of a character factor level. The first argument of the function is a pointer to LEVELC structure. Values for the LABELID and FACTORID elements must be provided. The corresponding observation unit for the LABELID and FACTORID is retrieved, along with the LEVELNO and LVALUE values. If fOpt = FIND\_FIRST (0), details of the first level value is retrieved, or the next available level value, of fOpt = FIND\_NEXT(1).

**Syntax**

Long **DMS\_getOunitLevelC**(*LEVELC,ouid,fopt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| LEVELC | DMS\_LEVELC | Input/Output | Address of a DMS\_LEVELC structure. |
| ounitid | Long | Output | The observation unit number being defined by the combination of factors and levels. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getEffectOunit**

DMS\_getEffectOunit extracts the observation units included in an effect representation.

**Syntax**

Long **DMS\_getEffectOunit**( *represno, ounitid, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| represno | Long | Input | The number of an effect representation. |
| ounitid | Long \* | Output | The observation unit number included in an effect representation. |
| fopt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getEffectRecord**

DMS\_getEffectRecord retrieves records of representation number, factor id and effect number of a study from the EFFECT table.

**Syntax**

Long **DMS\_getEffectRecord**(*studyid, effect, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | Long | Input | The identification number of a study. |
| effect | DMS\_EFFECT | Output | The address to the DMS\_EFFECT table. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getEffectRecordV**

DMS\_getEffectRecordV extracts the representation number, variate id and effect number of a study.

**Syntax**

Long **DMS\_getEffectRecord**(*studyid, effect, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | Long | Input | The identification number of a study. |
| Effect | DMS\_EFFECT | Output | The address to the DMS\_EFFECT table to contain the representation number, the variate id and effect number. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getEffect**

DMS\_getEffect retrieves the effect representations of a study.

**Syntax**

Long **DMS\_getEffect**(*studyid, effectid, repres,, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | Long | Input | The identification number of a study. |
| effectid | Long \* | Output | The identification number of an effect. |
| repres | Long \* | Output | The identification number of an effect representation. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getFactorEffect**

DMS\_getFactorEffect retrieves the factors of an effect representation.

**Syntax**

Long **DMS\_getFactorEffect**( *effect, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Effect | DMS\_EFFECT | Input/Output | The address to the DMS\_EFFECT structure. The representation number and effect id are passed to the function while the factor id is returned by the function. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getFactor**

DMS\_getFactor retrieves the factors of a given study. If the id of the factor label is specified, the function retrieves its record.

**Syntax**

**Long DMS\_getFactor( factor, fOpt)**

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Factor | DMS\_FACTOR | Input/Output | The address to the DMS\_FACTOR structure to contain the record of a factor label. The study id or the label id is a parameter passed to the function. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getFactorLabel**

DMS\_getFactorLabel retrieves the labels of a given factor id.

**Syntax**

Long **DMS\_getFactor**( *factor, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Factor | DMS\_FACTOR | Input/Output | The address to the DMS\_FACTOR structure to contain the record of a factor label. The factor id is a parameter passed to the function. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getVariate**

DMS\_getVariate retrieves the variates a given study. If the id of the variate is specified, the function gets its record.

**Syntax**

Long **DMS\_getVariate**( *variate, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| variate | DMS\_VARIATE | Input/Output | The address to the DMS\_VARIATE structure to contain the record of a variate. The id of the study or variate is a parameter passed to the function. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_getDMSAttr**

DMS\_getDMSAttr retrieves additional attributes of a given DMS entity. The type of entity or table name (e.g. STUDY, FACTOR,VARIATE) is provided in the TABLE\_NAME element of the DMS\_DMSATTR structure. The ATTRIBUTE\_TYPE element contains the id of the attribute type (e.g. Description), which is defined in the UDFLDS table. The record id of the entity in a DMS table (e.g. LABELID, VARIATID) is stored in the RECORDID\_IN\_TABLE element.

**Syntax**

Long **DMS\_getDMSAttr**( *attr, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| attr | DMS\_DMSATTR | Input/Output | The address to the DMS\_DMSATTR structure to contain the attribute of DMS entity. The type of entity, type of attribute and the specific id of the entity are parameters passed to the function. |
| fOpt | Long | Input | Either FIND\_FIRST (0) or FIND\_NEXT (1) |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS UPDATE FUNCTIONS**

**DMS\_updateDataC**

DMS\_updateDataC changes the character data value associated to the given ounitid and variatid.

**Syntax**

Long **DMS\_updateDataC**( *DataC*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| DataC | DMS\_DATAC | Input/Output | The address to the DMS\_DATAC structure that contains the new data value and the variatid and ounitid |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_updateDataN**

DMS\_updateDataN changes the numeric data value associated to the given ounitid and variatid.

**Syntax**

Long **DMS\_updateDataN**( *DataN*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| DataN | DMS\_DATAN | Input/Output | The address to the DMS\_DATAN structure that contains the new data value and the variatid and ounitid |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS DELETE FUNCTION**

**DMS\_deleteStudy**

DMS\_deleteStudy deletes all information about a study stored in the local DMS database.

**Syntax**

long **GMS\_deleteStudy**(*studyid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *studyid* | Long | input | the ID of the study to be deleted |

**Returns**

GMS\_SUCCESS, GMS\_ERROR

**MISCELLANEOUS FUNCTIONS**

**DMS\_existCombination**

DMS\_existCombination determines if a combination of factor levels exists or not in a study. It retrieves the observation unit which is defined by the array of factor ids and level numbers. It returns DMS\_SUCCESS if the combination exists.

**Syntax**

Long **DMS\_existCombination** (*studyid, arrLev, ounitid, nArr*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Studyid | Long | Input | The specific study where to search. |
| ArrLev | \*source | Input | The address of the first element of the array of factor ids and level numbers |
| Ounitid | Long | Output | The observation unit number being defined by the combination of factors and levels. |
| NArr | Long | Input/ Output | The number of factors in the array. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_existEffect**

DMS\_existEffect determines if there is an effect representation defined by the given list of factors. It retrieves the representation number and effect number if it exists and returns DMS\_SUCCESS.

**Syntax**

Long **DMS\_existEffect** (*lstfactor, nfactor, represno, effectid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| lstfactor | Long\* | Input | The address to the array of factors. |
| nfactor | Long | Input | The number of factors. |
| represno | Long\* | Output | The number of the representation defined by the given list of factors. |
| effectid | Long\* | Output | The number of the effect. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA (0)

**DMS\_minOunitid**

DMS\_minOunitid gets the minimum number of observation unit existing in the database. This function is primarily used in loading data to DMS.

**Syntax**

Long **DMS\_minOunitid**(*ounitid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ounitid | long | Output | The minimum number of observation unit existing in the database. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1), DMS\_NO\_DATA(0)

**DMS\_deleteStudy**

DMS\_deleteStudy deletes all records associated to the specified studyid.

**Syntax**

Long **DMS\_deleteStudy**(*studyid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| studyid | long | Output | The identification number of the study to be deleted. |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_deleteDataC**

DMS\_deleteDataC deletes a record from the DATA\_C table based on the specified ounitid and variateid.

**Syntax**

Long **DMS\_deleteDataC**(*ounitid, variateid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ounitid | long | Input | The observation unit of the record to be deleted. |
| variateid | long | Input | The variateid of the record to be deleted |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**DMS\_deleteDataN**

DMS\_deleteDataN deletes a record from the DATA\_N table based on the specified ounitid and variateid.

**Syntax**

Long **DMS\_deleteDataN**(*ounitid, variateid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| ounitid | long | Input | The observation unit of the record to be deleted. |
| variateid | long | Input | The variateid of the record to be deleted |

**Returns**

DMS\_SUCCESS(1), DMS\_ERROR(-1)

**IMS DLL FUNCTIONS**

The IMS\_openDatabase function should be called first before calling any function mentioned in this section. The different data structures used by the functions are discussed in section 2.

**ADD Functions**

**IMS\_addLot**

IMS\_addLot adds a new LOT record in the LOT table of the inventory database. LOTID is automatically assigned by the function. EID needs to exist already in the GERMPLSM table. If it doesn’t, the function returns IMS\_INVALID\_ID(-9).

**Syntax**

Long **IMS\_addLot(LOT)**

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotData* | IMS\_LOT | Input | Address of the IMS\_LOT structure that contains the information to be added to the database. |

**Returns**

IMS\_SUCCESS(1), IMS\_INVALID\_ID(-9), IMS\_ERROR(-1)

**IMS\_addTransaction**

IMS\_addTransaction adds a new transaction record in the Transactions table. TRNID is automatically assigned by the function. LOTID needs to exist in the LOT table. If it doesn’t, the function returns IMS\_INVALID\_ID (-9).

**Syntax**

long **IMS\_addTransaction('''***TRANSACTION'''***)**

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *transactionData* | IMS\_TRANSACTION | Input | Address of the IMS\_TRANSACTION structure that contains the information to be added to the database. |

**Returns**

IMS\_SUCCESS(1), IMS\_INVALID\_ID(-9), IMS\_ERROR(-1)

**DATA RETRIEVAL FUNCTIONS**

**IMS\_findLotRecord**

IMS\_findLotRecord retrieves data from the Lot table for a given

* *Etype*
* *Or etype and eid*
* *Or etype and locid*
* *Or etype and eid or locid*

Values specified in the ETYPE, EID, and LOCID fields of the IMS\_LOT structure.

If no records match the given criteria, the function returns IMS\_NO\_DATA .

**Syntax**

Long IMS\_findLotRecord ( *lotData, fOpt)*

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotData* | IMS\_LOT | Input/Output | Address of a IMS\_Lot record |
| *fOpt* | long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

IMS\_SUCCESS, IMS\_ERROR, NO\_DATA

**IMS\_findTransactionRecord**

IMS\_findTransactionRecord retrieves transaction records from the Transaction Table for a specified LOTID value.

**Syntax**

Long IMS\_findTransactionRecord (*lotid, transactionData, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *Lotid* | long | Input | The TRNID to search |
| *transactionData* | IMS\_TRANSACTION | Output | Address of a IMS\_Transaction record |
| *fOpt* | long | Input | FIND\_FIRST or FIND\_NEXT |

**Returns**

IMS\_SUCCESS, IMS\_ERROR, NO\_DATA

**IMS\_getTransactionReserve**

IMS\_getTransactionReserve retrieves all reserve transaction records from the Transaction Table. If the flagAll is not 1, then only transactions reserved by the current user are retrieved.

**Syntax**

Long IMS\_getTransactionReserve (*flagAll, transactionData,LotData, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *flagAll* | long | Input | 1 (all reserve) or 0 (only reserve done by the current user) |
| *transactionData* | IMS\_TRANSACTION | Output | Address of a IMS\_Transaction record |
| *lotData* | IMS\_LOT | Output | Address of a IMS\_Lot record |
| *fOpt* | long | input | FIND\_FIRST or FIND\_NEXT |

**Returns**

IMS\_SUCCESS, IMS\_ERROR, NO\_DATA

**IMS\_getTransactionDeposit**

IMS\_getTransactionDeposit retrieves all deposit transaction records from the Transaction Table. If the flagAll is not 1, then only transactions deposited by the current user are retrieved.

**Syntax**

Long IMS\_getTransactionDeposit (*flagAll, transactionData,LotData, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *flagAll* | long | Input | 1 (all deposit) or 0 (only deposit done by the current user) |
| *transactionData* | IMS\_TRANSACTION | Output | Address of a IMS\_Transaction record |
| *lotData* | IMS\_LOT | Output | Address of a IMS\_Lot record |
| *fOpt* | long | input | FIND\_FIRST or FIND\_NEXT |

**Returns**

IMS\_SUCCESS, IMS\_ERROR, NO\_DATA

**IMS\_getReserveByPerson**

IMS\_getReserveByPerson retrieves all reserve transaction made by a particular requestor. The PERSONID of the requestor is stored in the IMS\_TRANSACTION data structure (transactionData).

**Syntax**

Long IMS\_getReserveByPerson (*transactionData, LotData, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *flagAll* | long | Input | 1 (all reserve) or 0 (only reserve done by the current user) |
| *transactionData* | IMS\_TRANSACTION | Output | Address of a IMS\_Transaction record |
| *lotData* | IMS\_LOT | Output | Address of a IMS\_Lot record |
| *fOpt* | long | input | FIND\_FIRST or FIND\_NEXT |

**Returns**

IMS\_SUCCESS, IMS\_ERROR, NO\_DATA

**IMS\_getDepositByPerson**

IMS\_getDepositByPerson retrieves all deposits done by a particular person. The PERSONID is stored in the IMS\_TRANSACTION data structure (transactionData).

**Syntax**

Long IMS\_getDepositByPerson (*transactionData, LotData, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *flagAll* | long | Input | 1 (all reserve) or 0 (only reserve done by the current user) |
| *transactionData* | IMS\_TRANSACTION | Output | Address of a IMS\_Transaction record |
| *lotData* | IMS\_LOT | Output | Address of a IMS\_Lot record |
| *fOpt* | long | input | FIND\_FIRST or FIND\_NEXT |

**Returns**

IMS\_SUCCESS, IMS\_ERROR, NO\_DATA

**IMS\_getBalanceOfLot**

This function retrieves the total sum of the transaction quantities for a given lot.

**Syntax**

Long **IMS\_reportBalance** (*lotid, sumTrans*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | input | The lot |
| *sumTrans* | Double | output | The sum of the transaction quantities of the lot |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**UPDATE FUNCTIONS**

**IMS\_setTransaction**

This function changes the value of any field in of the IMS\_Transaction Table except LOTID and TRANSID given TRANSID. The status of the transaction can be updated by setting the TRNSTAT field to 1 (committed) or to 9 (cancelled) .

**Syntax**

Long **IMS\_setTransaction** (*transactionData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *transactionData* | IMS\_TRANSACTION | Input | Address of the IMS\_TRANSACTION data structure |

**Returns**

IMS\_SUCCESS, IMS\_ERROR

**IMS\_setLot**

This function changes the value of any field in of the IMS\_LOT Table except LOTID which is the given parameter.

**Syntax**

Long **IMS\_setLot** (*lotData*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotData* | IMS\_LOT | Input | Address of the IMS\_LOT data structure |

**Returns**

IMS\_SUCCESS, IMS\_ERROR

**REPORT FUNCTIONS**

**IMS\_reportAllTransaction**

This function retrieves all uncommitted transactions which include commitment date and quantity of the transaction, location, scale and comment about its lot.

**Syntax**

Long **IMS\_reportAllTransaction** (*lotid, cmtdate, quantity, szLocName, szScale, szComment, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | Output | The lot of the transaction |
| *cmtdate* | Long | output | The commitment date of the transaction |
| *quantity* | Double | output | The quantity of the transaction |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *szComment* | Char \* | output | Comment about the lot |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportMaterialDist**

This function reports on all requested reserve (uncommitted withdrawal).

**Syntax**

Long **IMS\_reportMaterialDist** (*lotid, eid, quantity, szLocName, szScale, szComment, cmtdate, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | Output | The lot of the transaction |
| *eid* | Long | Output | The ID of the entity (e.g. GID) in the lot |
| *quantity* | Double | output | The quantity of the transaction |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *szComment* | Char \* | output | Comment about the lot |
| *cmtdate* | Long | output | The commitment date of the transaction |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportByRequestor**

This function reports on all requested reserves and withdrawals including the persons who requested for them. The first record is returned if fOpt = FIND\_FIRST and the succeeding record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportMaterialDist** (*lotid, eid, quantity, szLocName, szScale, szComment, cmtdate, lngStat, lngPerson, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | Output | The lot of the transaction |
| *eid* | Long | Output | The ID of the entity (e.g. GID) in the lot |
| *quantity* | Double | output | The quantity of the reserve or withdrawal |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *szComment* | Char \* | output | Comment about the lot |
| *cmtdate* | Long | output | The commitment date of the transaction |
| *lngPerson* | Long | output | The person who reserved or withdrew the amount |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportBalance**

This function retrieves the sum of the transaction, location and scale of all lots. The first record is returned if fOpt = FIND\_FIRST and the succeeding record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportBalance** (*lotid, sumTrans, szLocName, szScale, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | Output | The lot of the transactions |
| *sumTrans* | Double | output | The sum of the quantity of the transactions |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportEmptyShelves**

This function reports on the lot that have zero balance or the sum of committed transactions is zero. The first record is returned if fOpt = FIND\_FIRST and the succeeding record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportEmptyShelves** (*lotid, eid, szLocName, szScale, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | Output | The lot which has zero balance |
| *eid* | Long | Input | The identification (GID) of the entity asoociated with the lot |
| *szLocName* | Char \* | output | The location where the lot is originally stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 (FIND\_NEXT) for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportDormantEntries**

This function reports on the lots with non-zero actual balance that are stored on or earlier than the specified year. The first record is returned if fOpt = FIND\_FIRST and the succeeding record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportDormantEntries** (*minYear, lotid, eid, quantity, szLocName, szScale, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *minYear* | Long | Input | The year used as basis for dormant entries |
| *lotid* | Long | Input | The lot considered as dormant |
| *eid* | Long | Input | The identification (GID) of the entity asoociated with the lot |
| *quantity* | Double | Input | The actual balance or sum of committed transactions |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 (FIND\_NEXT) for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportMinimumAmount**

This function reports on the lots with actual balance (or sum of committed transactions) less than the given amount. The first record is returned if fOpt = FIND\_FIRST and the succeeding record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportMinimumAmount** (*minQty, lotid, eid, quantity, szLocName, szScale, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *minQty* | Long | Input | The amount used as basis |
| *lotid* | Long | Input | The lot where it is considered below the minimum balance |
| *eid* | Long | Input | The identification (GID) of the entity asoociated with the lot |
| *quantity* | Double | Input | The actual balance or sum of committed transactions |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 (FIND\_NEXT) for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportLotByEntity**

This function retrieves the lot, scale, location and ids for a given type of Entity (e.g. GERMPLASM).

**Syntax**

Long **IMS\_reportLotByEntity** (*szType, lotid, eid, szLocName, szScale, szComment, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *szType* | Char \* | Input | The type of Entity. Generally, it is "GERMPLASM". |
| *lotid* | Long | Output | The lotid of a particular instance of the given Entity |
| *eid* | Long | Output | The identification (GID) of a particular instance of the entity |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *szComment* | Char \* | output | Comment about the lot |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportLotByEntityGIDFrom**

This function reports on the lot, scale, location for a given type of Entity (e.g. GERMPLASM) and its identification (e.g. GID). The first match is returned if fOpt = FIND\_FIRST and the next matching record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportLotByEntityGIDFrom** (*szType, lotid, eid, szLocName, szScale, szComment, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *szType* | Char \* | Input | The type of Entity. Generally, it is "GERMPLASM". |
| *lotid* | Long | Output | The lotid of a particular instance of the given Entity |
| *eid* | Long | Input | The identification (GID) of the entity |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *szComment* | Char \* | output | Comment about the lot |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 (FIND\_NEXT) for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**IMS\_reportLotByEntityGIDRange**

This function reports on the lot, scale, location for a given type of Entity (e.g. GERMPLASM) and range of its IDs (e.g. GID). The first match is returned if fOpt = FIND\_FIRST and the next matching record is returned if fOpt = FIND\_NEXT

**Syntax**

Long **IMS\_reportLotByEntityGIDRange** (*szType, lotid, eidFrom, eidTo, szLocName, szScale, szComment, fOpt*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *szType* | Char \* | Input | The type of Entity. Generally, it is "GERMPLASM". |
| *lotid* | Long | Output | The lotid of a particular instance of the given Entity |
| *eid* | Long | Input | The identification (GID) of the entity |
| *szLocName* | Char \* | output | The location where the lot is stored |
| *szScale* | Char \* | output | The scale or measurement unit on how the lot is stored |
| *szComment* | Char \* | output | Comment about the lot |
| *fOpt* | Long | input | 0 (FIND\_FIRST) to get the first record, 1 (FIND\_NEXT) for succeeding records |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA(0)

**MISCELLANEOUS**

**IMS\_closeLot**

This function changes the STATUS field of the Lot Table to 1 "Closed". Lot records are not deleted from the table.

**Syntax**

Long **IMS\_closeLot** (*lotid*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *lotid* | Long | Input | The lotid to close. |

**Returns**

IMS\_SUCCESS, IMS\_ERROR

**IMS\_calculateBalance**

IMS\_calculateBalance gets all values from field TRNQTY of the Transaction Table, for a given LOTID, and adds up all the anticipated and committed values to get the available balance, and all the committed values to get the actual balance. Cancelled transactions are ignored

**Syntax**

Long **IMS\_calculateBalance**(*lotid, trnqty1, trnqty2*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| *Lotid* | long | input |  |
| *Trnqty1* | long | Output | The summation of the TRNQTY field values for anticipated and committed transactions |
| *Trnqty2* | long | Output | The summation of the TRNQTY field values for committed transactions only. |

**Returns**

IMS\_SUCCESS(1), IMS\_ERROR(-1), IMS\_NO\_DATA (0)

**LMS DLL FUNCTIONS**

The GMS\_openDatabase function mentioned in Section IV.A.3 should be called first before calling any functions mentioned in this section. The different data structures used by the functions are discussed in section 2.

**GMS\_findLocation**

GMS\_findLocation searches the location table for the first location with name which matches the string in the PREFERRED\_NAME field of the *Locn* argument when *fOpt*=FIND\_FIRST. It searches for the next match if *fOpt* is FIND\_NEXT. If the TYPE field of *Locn* is non zero, the search is restricted to locations with LOCATION TYPE which match its value. The search string may contain wild cards such as \_ and %.

**Syntax**

long **GMS\_findLocation**(*Locn, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Locn | GMS Location Structure | Input/ output | Address of a GMS\_Germplasm record |
| fOpt | long | input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GMS\_getLocation**

GMS\_getLocation will retrieve location information. The first argument of GMS\_getLocation is a pointer to a GMS\_Location structure (6.1.2f). If the LOCATION\_ID is not zero when the function is called then it returns the details for the corresponding location ID. If it is zero then the function returns details for the first location record if *fOpt* = FIND\_FIRST or for the next available location record if *fOpt* = FIND\_NEXT. Any call which fails to get location information returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getLocation**(*location ,fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| location | GMS\_Location\* | input/output | Address of a GMS\_Location structure |
| fOpt | Long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

If the function is called and there is a problem accessing the database, the function will return GMS\_ERROR. Any call which fails to get location information returns GMS\_NO\_DATA.

**GMS\_getLocation2**

GMS\_getLocation2 will retrieve location information for all locations matching specified criteria. The first argument of GMS\_getLocation2 is a pointer to a GMS\_Location structure (6.1.2f). All elements of the structure must be set to valid values or zeros or blanks before calling the function with parameter fOpt=FIND\_FIRST. This sets a search template restricting the search to records with fields matching the valid values. The first match is returned if *fOpt* = FIND\_FIRST and the next matching location is returned if *fOpt* = FIND\_NEXT

**Syntax**

long **GMS\_getLocation2**(*location ,fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| location | GMS\_Location\* | input/output | Address of a GMS\_Location structure |
| fOpt | Long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**Comments**

The principle of using GMS\_getLocation2 is similar to GMS\_getAttribute2. If the function is called and there is a problem accessing the database, the function will return GMS\_ERROR. Any call which fails to get location information returns GMS\_NO\_DATA.

**GMS\_addLocation**

GMS\_addLocation adds a new location to the local LOCATION table. Details of the location must be stored in appropriate elements of the *location* parameter (6.1.2f), and the LOCATION\_ID element must be set to zero before the call to GMS\_addLocation. The next available location number will be returned in the LOCATION\_ID element

New data is not committed to the local database until the function GMS\_commitData (6.8.1) has been called, and uncommitted data can be deleted from the buffer with the GMS\_rollbackData function (6.8.2)

**Syntax**

long **GMS\_addLocation**(*location*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| Location | GMS\_Location \* | input/output | Address of a GMS\_Location structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *location* structure (6.1.2f) must be properly populated with values before calling the function:

• LOCATION\_ID must be set to 0,

• PREFERRED\_NAME must be set to a null terminated ASCII string,

Other elements may be set if values are known. Otherwise they should be set to the defined nulls for each field

**BIBREF DLL FUNCTIONS**

The GMS\_openDatabase function mentioned in Section 3.1 should be called first before calling any functions mentioned in this section. The different data structures used by the functions are discussed in section IV.A.2.

**GMS\_getBibrefs**

GMS\_getBibrefs retrieves bibliographic reference information. The first argument is a pointer to GMS\_BIBREFS structure. If the referenece ID is not zero when the function is called then it returns the details for the corresponding reference ID. If it is zero then the function returns details for the first bibliographic reference record if *fOpt* = FIND\_FIRST or for the next available reference record if *fOpt* = FIND\_NEXT. If no bibliographic information is found, the function returns GMS\_NO\_DATA.

**Syntax**

long **GMS\_getBibrefs**(*bibref ,fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| bibref | GMS\_BIBREFS\* | input/output | Address of a GMS\_BIBREFS structure |
| fOpt | Long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GMS\_addBibrefs**

GMS\_addBibrefs adds a new record of bibliographic reference to the local BIBREFS table. Details of the bibliographic reference must be stored in appropriate elements of the *bibref* parameter, and the REFID element must be set to zero before the call to GMS\_addBibrefs. The next available bibliographic reference number will be returned in the REFID element

New data is not committed to the local database until the function GMS\_commitData (see Section 3.1) has been called, and uncommitted data can be deleted from the buffer with the GMS\_rollbackData function (see Section 3.1)

**Syntax**

long **GMS\_addBibrefs**(*bibref*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| bibref | GMS\_BIBREFS\* | input/output | Address of a GMS\_BIBREFS structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *bibref* structure must be properly populated with values before calling the function:

• REFID must be set to 0,

• All character fields must be null terminated ASCII string,

Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS DLL FUNCTIONS**

The GEMS\_openDatabase function mentioned in Section 3 should be called first before calling any functions mentioned in this section. The different data structures used by the functions are discussed in section 2.

**ADD FUNCTIONS**

**GEMS\_addGEMSName**

(GEMS\_NAME \*recName) GEMS\_addGEMSName adds a new record of names for GEMS entities to the local GEMS\_NAMES table. Details of the names must be stored in appropriate elements of the *recName* parameter, and the GNID element must be set to zero before the call to GMS\_addGEMSName. The next available name ID will be returned in the GNID element.

**Syntax**

long **GEMS\_addGEMSName**(*recName*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| recName | GEMS\_NAME\* | input/output | Address of a GEMS\_NAME structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *recName* structure must be properly populated with values before calling the function:

• GNID must be set to 0,

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addComponent**

GEMS\_addComponent adds a new record of component of a polymorphic detector to the GEMS\_COMP table. Details of the component must be stored in appropriate elements of the *data* parameter, and the CID element must be set to zero before the call to GMS\_addComponent. The next available component ID will be returned in the CID element.

**Syntax**

long **GEMS\_addComponent**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_COMPONENT\* | input/output | Address of a GEMS\_COMPONENT structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• CID must be set to 0,

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addMD**

GEMS\_addMD adds a record in GEMS\_MARKER\_DETECTOR table. Details of the marker detector must be stored in appropriate elements of the *data* parameter, and the MDID element must be set to zero before the call to GMS\_addMD. The next available component ID will be returned in the MDID element.

**Syntax**

long **GEMS\_addMD**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_MARKER\_DETECTOR\* | input/output | Address of a GEMS\_MARKER\_DETECTOR structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• MDID must be set to 0,

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addMV**

GEMS\_addMV adds a record of information on molecular variant names in GEMS\_MV table. Details of the molecular variant must be stored in appropriate elements of the *data* parameter, and the MVID element must be set to zero before the call to GEMS\_addMV. The next available ID will be returned in the MVID element.

**Syntax**

long **GEMS\_addMV**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_MV\* | input/output | Address of a GEMS\_MV structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• MVID must be set to 0,

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addLocus**

GEMS\_addLocus adds a record in GEMS\_LOCUS table. GEMS\_LOCUS table contains information on chromosome and location of a molecular variant. Details of the locus must be stored in appropriate elements of the *data* parameter, and the LOCUSID element must be set to zero before the call to GEMS\_LOCUS. The next available ID will be returned in the LOCUSID element.

**Syntax**

long **GEMS\_addLocus**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_LOCUS\* | input/output | Address of a GEMS\_LOCUS structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• LOCUSID must be set to 0,

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addPD**

GEMS\_addPD adds a record in GEMS\_PD table. The PD table the different combination of Marker Detector ID(MDID) and condition ID (condid). Details of this relationship must be stored in appropriate elements of the *data* parameter, and the PDID element must be set to zero before the call to GMS\_addPD. The next available ID will be returned in the PDID element.

**Syntax**

long **GEMS\_addPD**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_PD\* | input/output | Address of a GEMS\_PD structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• PDID must be set to 0,

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addPdComp**

GEMS\_addPdComp adds a record in GEMS\_PD\_COMP table. The GEMS\_PD\_COMP (Polymorphic Detector and Components) table serves as the intermediate table to break the many-to-many relationship between the protocols/conditions and marker detector. Details of this relationship must be stored in appropriate elements of the *data* parameter, and the PD\_COMP element must be set to zero before the call to GMS\_addPdComp. The next available ID will be returned in the PD\_COMP element.

**Syntax**

long **GEMS\_addPdComp**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_PD\_COMP\* | input/output | Address of a GEMS\_PD\_COMP structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**GEMS\_addProp**

GEMS\_addProp adds a record in GEMS\_PROPERTY table. This table defines the property, method and scale used by each component in the GEMS\_COMP table. These information must be stored in appropriate elements of the *data* parameter.

**Syntax**

long **GEMS\_addProp**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_PROP\* | input/output | Address of a GEMS\_PROP structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addScale**

GEMS\_addScale adds a record in GEMS\_SCALE table. Details of the scale are stored in the appropriate elements of the *data* parameter.

**Syntax**

long **GEMS\_addScale**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_SCALE\* | input/output | Address of a GEMS\_SCALE structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GEMS\_addMethod**

GEMS\_addMethod adds a record in GEMS\_METHOD table. Details of the method are stored in the appropriate elements of the *data* parameter.

**Syntax**

long **GEMS\_addMethod**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_METHOD\* | input/output | Address of a GEMS\_METHOD structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR , GMS\_NO\_ACCESS

**Comments**

The following fields of the *data* structure must be properly populated with values before calling the function:

• All character fields must be null terminated ASCII string

• Other elements may be set if values are known. Otherwise they should be set to the default values.

**GET FUNCTIONS**

**GEMS\_getGemsMID**

GMS\_getGemsMID retrieves the corresponding GEMS MID based on the marker name & object type . The function returns details of the first matched record if *fOpt* = FIND\_FIRST or of the succeeding matched record if *fOpt* = FIND\_NEXT.

**Syntax**

long **GEMS\_getGemsMID**(*gemsInfo ,fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| gemsInfo | GEMS\_INFO\* | input/output | Address of a GMS\_INFO structure |
| fOpt | Long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_getGemsMVID**

GEMS\_getGemsMVID retrieves the corresponding GEMS MVID based on the allele name, object type, & marker ID. The function returns details of the first matched record if *fOpt* = FIND\_FIRST or of the succeeding matched record if *fOpt* = FIND\_NEXT.

**Syntax**

long **GEMS\_getGemsMVID**(*gemsInfo ,fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| gemsInfo | GEMS\_INFO\* | input/output | Address of a GMS\_INFO structure |
| fOpt | Long | Input | Either FIND\_FIRST or FIND\_NEXT |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_getNameID**

GEMS\_getNameID gets the corresponding GOBJID from GEMS\_NAMES table based on the given GOBJTYPE and GNVAL. The function returns the retrieved ID in the GOBJID element of the *data* parameter.

**Syntax**

long **GEMS\_getNameID**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_INFO\* | input/output | Address of a GMS\_INFO structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_getProtocolID**

GEMS\_getProtocolID gets the corresponding GOBJID from GEMS\_NAMES table where GOBJTYPE= PROTOCOL and on the given GNVAL. The function returns the retrieved ID in the GOBJID element of the *data* parameter.

**Syntax**

long **GEMS\_getProtocolID**(*data, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_INFO\* | input/output | Address of a GMS\_INFO structure |
| fOpt | Long | input | FIND\_FIRST = 0 or FIND\_NEXT = 1 |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_getPdComp**

GEMS\_getPdComp add all cids in gems\_comp where condition is equal to *Condition* of the *pdCompInfo* parameter. All cids found will be added to GEMS\_PD\_COMP table with pd\_comp equal to the next available id.

**Syntax** long **GEMS\_getPdComp**(*pdCompInfo, fOpt*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| pdCompInfo | GEMS\_PD\_COMP\* | input/output | Address of a GMS\_PD\_COMP structure |
| fOpt | Long | input | FIND\_FIRST = 0 or FIND\_NEXT = 1 |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_findPID**

GEMS\_findPID gets the corresponding PID from GEMS\_PROP table given the property, scale, method and property group. The function returns the retrieved ID in the PID element of the *data* parameter.

**Syntax**

long **GEMS\_findPID**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | PROP\_INFO\* | input/output | Address of a PROP\_INFO structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_findPdid**

gems\_pd.condid = ? AND gems\_pd.mdid = GEMS\_findPdid gets the corresponding PDID from GEMS\_PD table given CONDID and MDID. The function returns the retrieved ID in the PDID element of the *data* parameter.

**Syntax**

long **GEMS\_findPdid**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_PD\* | input/output | Address of a GEMS\_PD structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_findPropID**

GEMS\_findPID gets the corresponding PID and PROPID from GEMS\_PROP table given the property name and property group. The function returns the retrieved IDs in the appropriate elements of the *data* parameter.

**Syntax**

long **GEMS\_findPID**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_PROP\* | input/output | Address of a GEMS\_PROP structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_findScale**

GEMS\_findScale gets the corresponding SCALEID from GEMS\_SCALE table given the scale name and property id. The specified information should be stored in the appropriate elements of the *data* parameter. The function returns the retrieved ID in the SCALEID element of the parameter.

**Syntax**

long **GEMS\_findScale**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_SCALE\* | input/output | Address of a GEMS\_SCALE structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA

**GEMS\_findMethod**

GEMS\_findMethod retrieves a record of the method from GEMS\_METHOD table given the name. Details of the information are stored in the appropriate elements of the *data* parameter.

**Syntax**

long **GEMS\_findMethod**(*data*)

**Arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Type** | **Use** | **Description** |
| data | GEMS\_METHOD\* | input/output | Address of a GEMS\_METHOD structure |

**Returns**

GMS\_SUCCESS, GMS\_ERROR, GMS\_NO\_DATA